Integrierte WWW-Anfragesichten

Diplomarbeit

Universität Rostock, Fachbereich Informatik

vorgelegt von Gohla, Rolf
geboren am 06.03.1973 in Schwerin

Betreuer: Prof. Dr. Andreas Heuer
Prof. Dr.-Ing. Peter Forbrig
Dr.-Ing. Holger Meyer
Dipl.-Inf. Gunnar Weber
Dipl.-Inf. Jürgen Schlegelmilch

Abgabedatum: 18.02.2000

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Integration mehrerer WWW-Datenquellen in eine gemeinsame Anfragesicht, um Interoperabilität zwischen den autonomen WWW-Datenquellen für einen bestimmten Einsatzzweck zu erreichen. Die Anbindung der WWW-Datenquellen erfolgt über HTML-Wrapper. Dazu werden Design-Kriterien für Wrapper aufgestellt und existierende Toolkits für die Wrapper-Implementation hinsichtlich ihrer Fähigkeiten und Eignung untersucht. Zur Steuerung der Kontroll- und Datenflüsse zwischen den Wrappern und anderen notwendigen Komponenten einer Anfragesicht wird der Einsatz eines einfachen Workflow-Management-Systems vorgeschlagen. Hierfür wird ein selbstentwickeltes Workflow-Management-System vorgestellt sowie auf die Grundlagen der Arbeitsvorgangs- und Workflow-Modellierung eingegangen.

Zum Beispiel-Szenario *Reiseverbindungen* erfolgt die Konzeption und prototypische Implementierung einer integrierten WWW-Anfragesicht auf Basis des Workflow-Mangement-Systems. Diese ermittelt Reiseverbindungen aus den Auskunftssystemen der angebundenen Verkehrsunternehmen, im Beispiel sind das die Deutsche Bahn AG und der Fluganbieter Traveloverland.

Abstract

This master thesis deals with the integration of several WWW data sources into a common query view, in order to achieve interoperability for a special purpose between the autonomous data sources. The information retrieval from a WWW source is made by a HTML wrapper. In addition some design criteria are set up for a wrapper and existing toolkits used for wrapper implementation are examined in regard to their abilities and suitability. The use of a simple workflow management system is suggested, which controls the control flows and information flows between the wrapper applications and other necessary components of a query view. For this a self-developed workflow management system is presented. Therefore it is essential to deal with the basics of the workflow modelling.

The conception and prototypical implementation of an integrated view based on the workflow management system are described for the example scenario "travel connections". Such a integrated view creates travel connections consisting of data, which were retrieved from travel information systems of several transportation companies. In the example scenario will be queried the Deutsche Bahn AG and the flight provider Traveloverland.

CR-Klassifikation

H.3.3 Information Search and Retrieval

H.4 Information Systems Applications

H.2.8 Database Applications

Schlüsselworte

HTML, Wrapper, semi-strukturierte Daten, Informations-Retrieval, Arbeitsvorgangsmodellierung, Workflow-Management, Datenbankanwendung, W4F, JEDI, WebL, WWW

Inhaltsverzeichnis

Kapitel 1

Einführung

In den letzten Jahren ist das World Wide Web (WWW) explosionsartig gewachsen. Weltweit stellen viele Firmen, Behörden, Forschungseinrichtungen und Privatpersonen aus unterschiedlichsten Gründen Informations- und Serviceangebote unter einer eindeutigen Adresse bereit. Diese reichen von einfachen Textseiten bis hin zu komplexen formularbasierten Schnittstellen für darunterliegende Datenbanken. Darüber hinaus entwickelt sich das WWW zu einer Erstveröffentlichungquelle, das heißt, immer mehr Informationen erscheinen zuerst im WWW oder sogar nur dort. Durch die Schnelligkeit der elektronischen Informationsübermittlung wird eine Steigerung der Aktualität ermöglicht, die von hoher Bedeutung für den Nutzer sein kann, z. B. die Echzeitübertragung von Börsendaten. Die Unabhängigkeit der Informationsübermittlung vom Typ der verbreiteten Information (z. B. Text, Ton, Bilder, Videos) sichert ein breites Einsatzgebiet. Mit der voranschreitenden technischen Entwicklung erfolgt eine Verlagerung der Bedeutung von den klassischen Medien wie Zeitung, Buch, Fernsehen, Rundfunk hin zum WWW.

Ein Nutzer auf der Suche nach Informationen zu einem Thema steht nun vor mehreren Problemen. Als erstes muß er Anbieter[1] finden, deren Angebote relevante Daten enthalten könnten. Nach der manuellen Sichtung der Inhalte einiger Anbieter wählt er die Daten von Interesse aus. Basierend auf den gefundenen Daten werden weitere Interessensbereiche einbezogen, nach denen nun wieder gesucht wird. Letztendlich hat der Nutzer eine Menge von Daten gesammelt, die von unterschiedlichen Anbietern stammen bzw. aus Daten von unterschiedlichen Anbietern berechnet wurden. Da diese Anbieter autonom sind, muß er sich an die jeweilige Nutzerführung anpassen sowie Konflikte zwischen Datenformaten,

[1] Im weiteren Text wird auch von WWW-Datenquellen gesprochen.

Mehrdeutigkeiten usw. selbst auflösen. Diese Arbeitsweise ist unter dem oft gebrauchten Modewort *Surfen* zu verstehen.

Zur Aktualisierung der gesammelten Daten muß man die Arbeitsschritte erneut per Hand ausführen, was sehr mühsam sein kann. Eine große Erleichterung wäre dagegen eine vollständige oder weitgehende Automatisierung dieses Vorgangs. Genau das sollen *integrierte Web-Anfragesichten* leisten. Neben Zeitersparnis und Komfortsteigerung gewinnt eine solche Informationsquelle auch für jene Nutzer an Bedeutung, die sich, mangels Kenntnissen, nicht selbst diese Daten hätten erschließen können. Existierende Beispiele sind Preisagenten und Metasuchmaschinen.

Die Arbeit beschäftigt sich mit der Realisierung von integrierten Web-Anfragesichten. Dabei lassen sich zwei wesentliche Bereiche herausstellen. Zum einem ist das die Schaffung von Zugriffsschnittstellen für die WWW-Datenquellen, die sogenannten *Wrapper*. Kapitel 2 behandelt diese Thematik detailliert. Unter anderem wird speziell auf das Wrapper-Toolkit *W4F* zur Erzeugung von HTML-Wrappern eingegangen. Zum anderen muß es Kontroll- und Datenflußkomponenten geben, die die Interoperabilität zwischen den Wrappern und zwischen anderen Komponenten herstellen. Eine integrierte Anfragesicht sollte modular aufgebaut sein, um die einzelnen Teilaufgaben voneinander zu trennen. Das sichert Erweiterbarkeit und Übersichtlichkeit, erleichtert die Wartung und ermöglicht die Integration verschiedenster Komponenten zur Ausführung der notwendigen Teilaufgaben. Um diese Eigenschaften zu erreichen, bietet sich ein Workflow-Management-System zur Realisierung von integrierten WWW-Anfragesichten an. In der Studienarbeit [Goh99] wurde ein kleines Workflow-Management-System konzipiert und implementiert. Dieses Workflow-Management-System wird als Basis für integrierte Anfragesichten verwendet. Dazu muß auf die theoretischen Grundlagen der Workflow-Modellierung eingegangen werden, das geschieht in Kapitel 3. Das implementierte Mini-Workflow-Management-System selbst wird in Kapitel 4 vorgestellt. Es beschreibt die Konzeption und die Funktionsweise des Systems. Als praktische Anwendung für eine integrierte Web-Anfragesicht wird in Kapitel 5 das Beispiel-Szenario *Reiseverbindungen* vorgestellt.

Kapitel 2

Automatisches Abfragen von Datenquellen des WWW

Dieses Kapitel beschäftigt sich mit der automatischen Gewinnung von Daten aus WWW-Datenquellen. Ein konzeptionelles Problem in der Herangehensweise stellt sich allerdings gleich am Anfang dar. Der Zugriff auf die Daten einer WWW-Datenquelle ist für den menschlichen Nutzer konzipiert, jetzt soll aber mit einer Software, dem Wrapper, auf die Daten zugegriffen werden. Das heißt, zur automatisierten Datengewinnung wird eine Mensch-Maschinen-Schnittstelle benutzt. Ein solcher Lösungsansatz ist in der Regel mit erheblichen Mängeln belastet, da eine Maschine den Menschen ersetzen soll, ohne daß sich die Arbeitsumgebung ändert. Es gibt aber momentan keine anderen Möglichkeiten, Daten von WWW-Datenquellen abzufragen.

In Abschnitt 2.1 erfolgt der Klärung des Begriffes *Wrapper*. Anschließend werden im Abschnitt 2.2 Konzepte für das Wrapper-Design beschrieben. Der Hauptabschnitt 2.3 des Kapitels beschäftigt sich mit mehreren Toolkits, die zur Programmierung von Wrappern dienen. Dabei wird auf deren Konzepte und deren Handhabung eingegangen, und es werden abschließend die Eigenschaften der Toolkits gegenübergestellt. Zuletzt werden in Abschnitt 2.4 Probleme dargelegt, die allgemein beim Abfragen von WWW-Datenquellen auftreten und den Einsatzbereich von Wrapper-Systemen einschränken.

2.1 Aufgabe eines Wrappers

Zur Eingrenzung der Aufgabe erfolgt eine Beschränkung auf eine Teilmenge des WWW. Es werden nur solche Datenquellen berücksichtigt, die ihre Daten in der

Seitenbeschreibungssprache HTML ausgeben und zur Kommunikation das Übertragungsprotokoll HTTP benutzen.

Was ist nun ein Wrapper ?

Allgemein versteht man unter dem Begriff *Wrapper* eine Software zur Konvertierung von Daten und Anfragen zwischen zwei Datenmodellen.

Ein *Wrapper für eine WWW-Datenquelle* extrahiert im HTML-Dokument implizit gespeicherte Daten und konvertiert diese in explizit gespeicherte Daten eines Datenmodells. Das HTML-Dokument ist Ergebnis einer Anfrage an die WWW-Datenquelle mittels des HTTP-Protokolls[1].

Um nicht zuweit auszuholen, folgt hier nur eine Kurzerläuterung des HTTP-Protokolls. Ein WWW-Client verbindet sich mit einem über eine URL (**U**nifed **R**equest **L**ocator) adressierten HTTP-Server unter Angabe der Übertragungsmethode GET oder POST für zu übermittelnde Parameter. Die Parameter haben die Form von Name-Wert-Paaren. Der HTTP-Server liefert als Ergebnis einer erfolgreichen Anfrage ein HTML-Dokument zurück. Außerdem liefert der Server immer eine Menge von HTTP-Parametern zurück, über die dem WWW-Client Fehlerzustände, Umleitungen, Authorisationsanforderungen usw. mitgeteilt werden und die er behandeln können sollte.

Da die zu extrahierenden Daten implizit im HTML-Dokument gespeichert sind, muß erst eine Zielstruktur und der Typ der Daten ermittelt werden. Zur Bestimmung des Datenmodells hilft die Berücksichtigung der HTML-Struktur und der Struktur des Textes. Desweiteren werden natürlich Verfahren zur Extraktion der Daten aus dem HTML-Dokument benötigt. Dieses *Reverse-Engineering* ist Aufgabe des Wrapper-Entwicklers.

2.2 Design von Wrappern

Dieser Abschnitt beschäftigt sich mit Entwurfskriterien für Wrapper und stellt zwei Design-Konzepte vor.

Da sich das Layout von HTML-Seiten im WWW recht häufig ändert, wäre es wünschenswert, wenn der Wrapper an die Änderungen leicht anpassbar ist (*Änderungsfreundlichkeit*). Außerdem sollte er an kleineren Layout-Änderungen nicht unbedingt scheitern, sondern wenigstens einen Großteil der gewünschten Daten extrahieren (*Robustheit*). Sehr nützlich wäre auch, eine fertige Extraktionslösung bei einem anderen Wrapper wiederzuverwenden (*Wiederverwendbarkeit*). Der

[1] Eine genaue Beschreibung von HTTP und HTML findet der Leser in [MK97].

Einsatz auf anderen Plattformen (*Portierbarkeit*) und der nachträgliche Wechsel der Programmiersprache (*Implementationsunabhängigkeit*) sind ebenfalls Kriterien für das Wrapper-Design.

2.2.1 Klassisches Design

Unter einem *klassischen Wrapper* versteht man ein Programm in einer Programmiersprache, bei dem die Extraktionslösung und die Zieldatenstruktur Bestandteil des Programmcodes sind. Das heißt, es gibt keine Trennung zwischen dem Programmcode für den Programmablauf und dem Programmcode für die Datenextraktion und die Abbildung auf die Zieldatenstruktur. Dadurch entsteht ein hoher Aufwand bei nötigen Anpassungen des Wrappers, da erst der Programmcode analysiert werden muß. Die Wahrscheinlichkeit, daß Seiteneffekte durch die Änderung auftreten, ist aufgrund der fehlenden Übersichtlichkeit hoch. Die Wiederverwendbarkeit einer Extraktionslösung wird stark behindert, da der Programmcode für die Extraktion nur schwer vom Programm getrennt werden kann. Je nach Verfügbarkeit der gewählten Programmiersprache auf anderen Plattformen könnte die Portierbarkeit eingeschränkt sein. Ein nachträglicher Wechsel der Programmiersprache ist nicht möglich, der Wrapper müßte in diesem Fall neu implementiert werden.

2.2.2 Design-Richtlinien des W4F-Toolkits[2]

Um den oben genannten Kriterien an einen Wrapper besser zu genügen, wurden von den W4F-Entwicklern Design-Richtlinien aufgestellt und im W4F-Toolkit umgesetzt. Da diese grundlegende, allgemeingültige Konzepte für die Wrapper-Implementierung vermitteln, werden sie nachfolgend beschrieben.

- Es erfolgt die Verwendung einer modularen dreistufigen Schichtenarchitektur, um Unabhängigkeit und Wiederverwendbarkeit zu bieten. Dabei stellen die Schichten die Teilaufgaben des Wrappers dar, im einzelnen sind das die *Retrieval-Schicht*, die *Extraktionsschicht* und die *Abbildungsschicht*.

  - In der Retrieval-Schicht erfolgt das Abfragen der WWW-Datenquelle, das Ergebnis ist das HTML-Dokument.
  - In der Extraktionschicht werden die Daten von Interesse aus dem HTML-Dokument extrahiert. Dabei sollte die benutzte Sprache die

[2]Entnommen aus [SA98b].

HTML-Struktur des Dokumentes und die Struktur der Text-Elemente ausnutzen können.

  - In der Abbildungsschicht erfolgt die Abbildung der extrahierten Daten auf eine Zieldatenstruktur, die dann weiter benutzt werden kann.

- Die einzelnen Schichten werden deklarativ spezifiziert. Jede wird durch eine Menge von Spezifikationsregeln beschrieben, dadurch gibt es eine eindeutige Semantik. Dies sichert Verständlichkeit, leichtere Wartbarkeit und Wiederverwendbarkeit. Die zur Implementation benutzte Programmiersprache kann ausgetauscht werden, die Spezifikation ändert sich dadurch nicht. Letztendlich ist die Spezifikationssprache leichter erlernbar, da diese eine viel kleinere Grammatik als eine vollständige Programmiersprache besitzt.

- Die Extraktion der Daten muß an deren Granularität anpaßbar sein. Damit ist gemeint, daß die Datenstruktur einerseits anhand der HTML-Struktur ermittelt wird, anderseits aber auch die Textstruktur innerhalb eines HTML-Tags auswertbar sein muß, z. B. eine mit Kommas getrennte Aufzählung. (Dies leisten beispielsweise reguläre Ausdrücke.)

- Es ist eine abstrakte Repräsentation des HTML-Dokumentes nötig. Dies ermöglicht eine eindeutige Identifikation der Elemente und eine einfachere Navigation im HTML-Dokument. Dadurch wird eine deklarative Spezifikation erst ermöglicht.

- Unterstützungswerkzeuge sollten bei der Erstellung und Wartung von Wrappern helfen, um unnötigen Aufwand zu vermeiden oder die Einarbeitungszeit zu verkürzen.

Im Abschnitt 2.3.1 wird die Wrapper-Implementierung unter Berücksichtigung dieser Design-Richtlinien mittels des W4F-Toolkits beschrieben.

2.3 Wrapper-Toolkits

In diesem Abschnitt werden drei Toolkits zur Implementierung von Wrappern für WWW-Datenquellen vorgestellt. Dabei handelt es sich um die Systeme W4F, JEDI und WebL, die alle drei frei verfügbar sind. Sie besitzen unterschiedliche konzeptuelle Ansätze, aber auch Gemeinsamkeiten. In Abschnitt 2.3.4 ist eine Gegenüberstellung der Toolkits zu finden.

2.3.1 W4F[3]

W4F bedeutet *WysiWyg Web Wrapper Factory* und stammt von der Universität Pennsylvania, USA und von der Telecom (E.N.S.T.) Paris, Frankreich.

Das Toolkit basiert auf Java und besteht aus mehreren Komponenten. Das sind im folgenden:

- Der *W4F-Parser* dient zur Analyse von HTML-Dokumenten. Es wird HTML 3.2 unterstützt. Der verwendete Parser ist fehlertolerant, da er fehlerhafte HTML-Seiten trotzdem teilweise analysieren kann.
- Der *W4F-Compiler* generiert eine Java-Klasse zu einer Wrapper-Spezifikation. Diese Java-Klasse kann in jedem Java-Programm benutzt werden. Durch Einsatz eines anderen Compilers wäre es möglich, ein Programm in einer anderen Zielsprache zu erzeugen, ohne daß sich die Wrapper-Spezifikation ändert. Die Abbildungsschicht des Wrappers ist davon allerdings ausgenommen, da sie die Daten auf ein konkretes Datenmodell abbildet.
- Das *W4F-Laufzeitmodul* erlaubt die Ausführung der generierten Java-Wrapper als selbständige Programme. Besonders beim Testen der Funktionsweise des Wrappers ist das eine große Hilfe.
- Zur Unterstützung des Nutzers existieren mehrere Werkzeuge oder sind in Vorbereitung. Diese laufen nur mit dem Internet-Explorer 4 oder höher und sind meiner Meinung nach nur für Einsteiger in W4F und HTML interessant.
  - Der *Formular-Wizard* hilft bei der Erstellung von Retrieval-Regeln für WWW-Seiten, die Formulare enthalten. Er gibt die wesentlichen Elemente eines HTML-Formulares aus.
  - Der *Extraktions-Wizard* hilft bei der Erstellung von Extraktionsregeln. Er gibt den identifizierenden Pfadausdruck für ein Textelement der HTML-Seite an, wenn der Nutzer ihn im Browser selektiert. Daraus leitet sich das *wysiwyg* (**w**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et) im Namen ab.
  - Mit dem *Abbildungs-Wizard* ist eine Hilfe für den Nutzer bei der Abbildung vom intern verwendeten Datenformat NSL auf nutzerdefinierte Datenstrukturen geplant.

Nach diesem kurzen Überblick erfolgt eine Beschreibung der Wrapper-Erstellung nach den in Abschnitt 2.2.2 genannten Design-Richtlinien des W4F-Toolkits.

[3]Die Informationen zu W4F stammen aus [SA98b] und [SA98a].

2.3.1.1 Prinzipielle Arbeitsweise eines W4F-Wrappers

In Abbildung 2.1 ist die prinzipielle Arbeitsweise eines W4F-Wrappers dargestellt. Ein Wrapper besteht aus drei Teilen, die der vorgestellten Schichtenarchitektur entsprechen. In der Retrieval-Schicht wird das durch die Retrieval-Regeln spezifizierte HTML-Dokument geladen und dient als Eingabe für den HTML-Parser, der daraus eine abstrakte Darstellung, den *HTML-Baum* oder *Analysebaum*, erstellt. Anhand dieses Analysebaums kann jedes Element des HTML-Dokumentes über einen Pfadausdruck eindeutig identifiziert werden. In der Extraktionsschicht erfolgt die Gewinnung der in den Extraktionsregeln spezifizierten Datenelemente des HTML-Baums. Diese werden in der intern verwendeten Datenstruktur *NSL* (***N**ested **S**tring **L**ist*) gespeichert. In der sich anschließenden Abbildungsschicht geschieht das Abbilden dieser Daten in NSL auf die vom Nutzer angegebene Zieldatenstruktur. Ergebnis ist dann ein Java-Objekt bzw. ein Java-Standarddatentyp. Im folgenden wird auf die einzelnen Schichten genauer eingegangen.

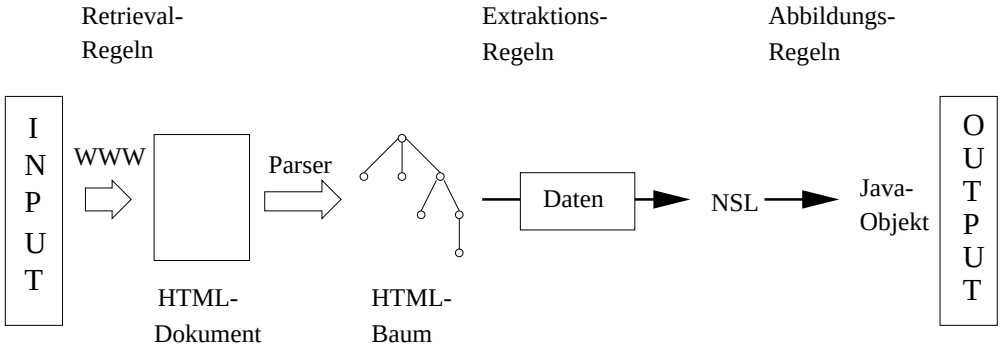

Abbildung 2.1: Prinzipielle Arbeitsweise eines W4F-Wrappers

2.3.1.2 Die Retrieval-Schicht

Durch die Spezifikation von Regeln in der Retrieval-Sprache[4] wird festgelegt, welche HTML-Seite mit welchen Parametern von welcher Web-Adresse mit welcher Übertragungsmethode (GET oder POST) des HTTP-Protokolls geladen wird. Das Holen des HTML-Dokumentes aus dem WWW geschieht für die restlichen Schichten des Wrappers vollkommen transparent, die Extraktionsschicht erhält als Eingabe das gewonnene HTML-Dokument.

[4]Die Grammatik ist in [w4f] zu finden.

Dies ist ein Beispiel für die Ermittlung eines HTML-Dokumentes mit der GET-Methode des HTTP-Protokolls. Die Web-Adresse (URL) ist bei dieser Regel variabel. Eventuelle Parameter werden in der URL übergeben.

```
// GET-Methode
get_document(String url)
{
  METHOD: GET;
  URL: $url$;
}
```

Nachfolgend steht ein Beispiel für die POST-Methode. Deren Parameter werden in Form von Name-Wert-Paaren hinter dem Token `PARAM:` angegeben. Dabei können die Werte variabel gehalten sein.

```
// POST-Methode
get_document(String keyword)
{
  METHOD: POST;
  URL: "http://www.amazon.com/exec/obidos/xxxx";
  PARAM: "keyword-query" = $keyword$, "mode" = "books";
}
```

Der Retrieval-Block im Wrapper-Quelltext beginnt mit dem Token `RETRIEVAL_RULES`. Dabei ist anzumerken, daß zwar mehrere Retrieval-Regeln in diesem Block spezifizierbar sind, aber nur eine zu einem bestimmten Zeitpunkt aufgerufen werden kann und damit immer nur ein HTML-Dokument der Extraktionsschicht übergeben wird.

2.3.1.3 Die Extraktionsschicht

Nach Abschluß der Retrieval-Schicht beginnt der HTML-Parser automatisch mit der Erstellung des Analysebaums zum gewonnenen HTML-Dokument. Die Zuordnung von HTML-Dokument und Analysebaum ist eineindeutig, d. h. dasselbe HTML-Dokument ist wieder aus dem Analysebaum ermittelbar und umgekehrt.

Der Analysebaum besteht aus drei Bausteinen. Das sind im einzelnen:

- Die *Wurzel*. Sie wird mit dem Namen `html` bezeichnet und ist der Startknoten im Analysebaum.
- *Interne Knoten*. Diese repräsentieren die geschlossenen HTML-Tags des HTML-Dokumentes und werden mit deren Namen bezeichnet. Geschlossene HTML-Tags sind alle HTML-Sprachelemente, die durch eine Start-

und Endmarke gebildet werden, z. B. `<table> ... </table>`. Sie kapseln Unterbäume, haben also Kinder und können Attribute besitzen.

- *Blätter*. Das sind Knoten, die zum einen die offenen HTML-Tags repräsentieren und deren Namen als Bezeichnung dienen. Zum anderen werden die Textstücke des HTML-Dokumentes als Blätter des Analysebaums abgebildet und mit dem Namen `PCDATA` benannt. Sie haben keine Kinder, können aber Attribute besitzen.

In Abbildung 2.2 auf der nächsten Seite ist zur Veranschaulichung der Analysebaum zu nachstehendem HTML-Dokument abgebildet:

```
<HTML>
 <HEAD><TITLE>Text</TITLE> </HEAD>
 <BODY>
  <H1>Text</H1>
  <IMG>
  <TABLE>
   <TR>
    <TH>Text<BR>Text</TH>
    <TH> ... </TH>
    <TH> ... </TH>
   </TR>
   <TR>
    <TD><IMG><A><B>Text</B></A><BR><IMG></TD>
    <TD> ... </TD>
    <TD> ... </TD>
   </TR>
   <TR>
    ...
   </TR>
  </TABLE>
 </BODY>
</HTML>
```

Die Identifikation der Elemente des Baumes sowie der Zugriff auf diese geschieht über Pfadausdrücke, die aus Knotennamen, Operatoren, Indizes bzgl. der Knotennamen und Attributnamen gebildet werden. Dabei geschieht die Vergabe der Indizes nach dem *left-depth-first-Prinzip*, d. h. es wird zuerst links absteigend im Baum nach einem vorgegebenen Knotennamen gesucht und das Auftreten gezählt. Bei Übereinstimmung von Index und Zähler wurde der gesuchte Knoten gefunden. Zu beachten ist, daß der Indexwert Null das erste Auftreten adressiert.

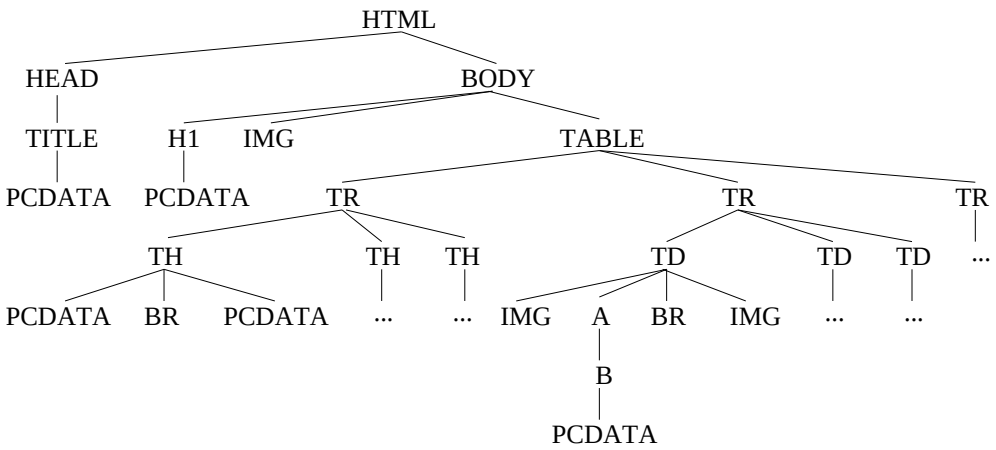

Abbildung 2.2: Beispiel eines HTML-Baums

Wurde der Index weggelassen, ist immer der Index Null gemeint. Zur Bildung der Pfadausdrücke stehen zwei Operatoren zur Verfügung, die aus unterschiedlichen Navigationsansätzen stammen.

- Der *Punkt-Operator* `.` nutzt die Baumhierarchie aus. Dabei wird hinter jedem Element im Pfadausdruck getrennt durch den Punkt-Operator ein unmittelbarer Kindknoten im Analysebaum angegeben. Damit läßt sich zu jedem Knoten genau ein Pfadausdruck angeben. Zur Verdeutlichung ein Beispiel:
  `html.head.title`
  `html.body.table[0].tr[1].td[0].img`

- Der *Pfeil-Operator* `->` orientiert sich am Fluß im Dokument. Ausgehend von einer Position im Analysebaum sucht er das nächste Auftreten des angegebenen Knotens unter Berücksichtigung eines eventuellen Indexes. Dabei arbeitet er über die Hierarchie des Baumes hinweg. Für einen Knoten lassen sich mehrere mit dem Pfeiloperator gebildete Pfadausdrücke angeben. Ein Beispiel:
  `html.body->img[1]`
  `html.body->img[0]->img[0]`

Als Index für ein Pfadelement können auch Index-Intervalle oder Wildcards verwendet werden, so daß mehrere Knoten das Ergebnis bilden. Beispielsweise liefert `html.body->a[*]` alle Anker des HTML-Dokumentes während `html.body->ul[2-4]->li[*]` alle Listenelemente der 3. bis 5. ungeordneten Liste des HTML-Dokumentes liefert.

Interne Knoten und Blätter können HTML-Attribute besitzen, während `PCDATA`-Blätter Textwerte als Attribute zugeordnet haben. In Tabelle 2.1 sind die Eigenschaften der Knotentypen zusammengefaßt. Mittels spezieller Zugriffsfunktio-

	Wurzel	Interne Knoten	Blätter	PCDATA
Kinder	x	x	-	-
HTML-Attribute	-	x	x	-
Textwert	-	-	-	x

Tabelle 2.1: Eigenschaften von Knoten des HTML-Baums

nen können diese abgefragt werden. Das geschieht durch Anhängen des Punkt-Operators und des Funktionsnamens an das letzte Element im Pfadausdruck. Folgende Zugriffsfunktionen stehen zur Verfügung:

- Die Funktion `txt` liefert den Textwert eines PCDATA-Blattes oder eine rekursive Verknüpfung der Textwerte im gesamten Unterbaum eines Knotens mit Kindern. Die Reihenfolge der Verknüpfung ist linksabsteigend zuerst.
  Beispiele:
  - `html.title.pcdata.txt` ermittelt den Titel des HTML-Dokumentes.
  - `html.body.table[0].tr[0].th[0].txt` ermittelt die Spaltenüberschrift der ersten Tabellenspalte.
- Die Funktion `src` liefert den HTML-Quelltext zu einem Knoten. Die Bildung des Quelltextes zu einem Unterbaum geschieht analog zur `txt`-Funktion.
  Beispiel:
  - `html.src` gibt den gesamten Quelltext des HTML-Dokumente aus.
- Die Funktion `getAttr(<html_attribut_name>)` gibt den Attributwert des angegebenen HTML-Attributes eines Knotens zurück. Als Sonderfall ist der Wurzel als Attribut der empfangene HTTP-Kopf zum HTML-Dokument zugeordnet .
  Beispiele:
  - `html.body->a[0].getAttr(href)` liefert die URL des ersten Links im HTML-Dokument.
  - `html.getAttr(url)` liefert die URL des HTML-Dokumentes.

- Die Funktion `numberOf(<knotenbezeichnung>)` zählt das Auftreten eines bestimmten Knotens in einem Unterbaum. Als Sonderfall zählt `numberOf(all)` alle Knoten eines Unterbaums.
  Beispiele:
  - `html.body.table[0].numberOf(tr)` gibt die Zeilenanzahl der ersten Tabelle zurück.
  - `html.numberOf(all)` ermittelt die Anzahl aller HTML-Tags im Dokument.

Die Anwendbarkeit der Funktionen auf die einzelnen Knotentypen ist in Tabelle 2.2 zusammengefaßt.

	Wurzel	Interne Knoten	Blätter	PCDATA
`txt`	x	x		x
`src`	x	x	x	x
`getAttr()`	x	x	x	-
`numberOf()`	x	x	-	-

Tabelle 2.2: Zugriffsfunktionen der Knotentypen

Die vorgestellten Pfadausdrücke benutzt man bei der Bildung der Extraktionsregeln, die in der Sprache *HEL* (***H**TML **E**xtraction **L**anguage*) spezifiziert werden. Die genaue Grammatik ist in [w4f] zu finden. Die Extraktionsregeln liefern als Ergebnis die Attributwerte bestimmter Knoten des HTML-Baums im intern verwendeten NSL-Format. Eine *NSL (**N**ested **S**tring **L**ist)* ist eine geschachtelte Liste von String-Listen beliebiger Tiefe. Der NSL-Datentyp ist folgendermaßen definiert:

$$NSL = null \mid string \mid listof(NSL)$$

Die Struktur einer NSL, das ist die Dimension, Tiefe und Zahl der Nestungen, wird allein durch die angegebenen Pfadausdrücke und deren Verknüpfung mit HEL-Operatoren bestimmt. Beispielsweise liefert die Extraktionsregel `urls = html.body->a[*].getAttr(href)` eine NSL der Form *listof(string)*, die Regel `array = html.body->ul[2-4]->li[*].txt` eine NSL der Form *listof(listof(string))*.

Weitere HEL-Sprachelemente sind die Operatoren `match`, `split` und `#`.

- Mit `match` und `split` wird versucht, Strukturen in Textstücken auszunutzen, die von der Struktur des HTML-Dokumentes nicht ausgedrückt werden. Sie arbeiten auf einer feineren Datengranularität und benutzen dazu reguläre Ausdrücke.

- Der fork-Operator # erlaubt die Aufspaltung eines Pfadausdruckes in mehrere Unterpfadausdrücke und gruppiert die Teilergebnisse der Unterpfade wieder zu einem Gesamtergebnis. Zum besseren Verständis ein Beispiel: Die Regel `links = html.body->a[*](.txt # .getAttr(href))` ermittelt alle Links in einem HTML-Dokument mit Beschreibung und URL. Die Struktur der NSL `links` ist *listof(listof(string, string))*. Der fork-Operator ist sehr nützlich, da er die Struktur zusammengehörender Daten im HTML-Dokument in die Struktur des Ergebnisses überführen kann. Da die mehrmalige Anwendung des Operators in einem Pfadausdruck erlaubt ist, können so irreguläre Listen enstehen. Das heißt, die Elemente der NSL weisen unterschiedliche Dimensionen und Schachtelungstiefen auf. Es ist Aufgabe der Abbildungsschicht, solche Listen verarbeiten zu können.

Durch die Angabe von Bedingungsklauseln kann das zu ermittelnde Ergebnis weiter präzisiert werden. Dazu werden im Pfadausdruck statt festen Indizies Variablen eingesetzt, deren Wert jeweils durch eine Bedingung bestimmt wird. Für jede eingeführte Variable muß genau eine Bedingung angegeben sein. Die Bedingungsklausel wird durch das Schlüsselwort `WHERE` eingeleitet, und die Bedingungen dürfen nur konjunktiv miteinander verknüpft sein. Die Bedingungen sind nur über den Knotenattributen formulierbar, nicht auf den Knoten selbst. Es existieren eine Reihe von Vergleichsoperatoren für numerische Werte und Zeichenketten wie "`=`" (gleich), "`!=`" (ungleich), "`=~`" (enthalten sein), "`!=~`" (nicht enthalten sein). Festzuhalten ist, daß die Ergebnisstruktur allein durch den Pfadausdruck bestimmt wird und nicht durch die Bedingungen.

Zum Verständnis ein Beispiel:
In Abbildung 2.3 auf der nächsten Seite ist eine Währungsumrechnungstabelle aus einem HTML-Dokument abgebildet, ein Service, den der Internet-Dienstleister Yahoo! anbietet. Nachstehende Extraktionsregel ermittelt den aktuellen Kurs zwischen Dollar und D-Mark. Dabei wird die Zeile und Spalte variabel gehalten, um von Änderungen an der Tabelle nicht beeinflußt zu werden, z. B. das Hinzufügen einer neuen Währung. Die erste Spaltenüberschrift, die den String "`DM`" enthält, legt die Variable `j` fest, der erste Zeilenkopf, der "`U.S.`" enthält, die Variable `i`.

```
value= html.body->table[1].tr[i].td[j].txt
WHERE  html.body->table[1].tr[0].th[j].txt =~ "DM"
AND    html.body->table[1].tr[i].th[0].txt =~ "U.S.";
```

Der Extraktions-Block im Wrapper-Quelltext beginnt mit dem Token `EXTRACTION_RULES`. Danach folgen die Extraktionsregeln, wobei die Variable auf der linken Seite der Extraktionsregel im Abbildungsblock deklariert sein muß, falls sie auf einen anderen Datentyp als das intern verwendete NSL-Format abgebildet werden soll.

Currency	U.S. $	Aust $	U.K. Pound	DMark	FFranc
U.S. $	1	0.6244	1.657	0.6024	0.1796
Aust $	1.602	1	2.654	0.9648	0.2877
U.K. Pound	0.6034	0.3768	1	0.3635	0.1084

Abbildung 2.3: Währungsumrechnungstabelle für Beispiel-Extraktionsregel

2.3.1.4 Die Abbildungsschicht

In der Abbildungsschicht kann der Nutzer angeben, auf welchen Zieldatentyp die durch die Extraktionsregeln gewonnene interne NSL-Darstellung abgebildet werden soll. Dabei kann eine Abbildung nur für jeweils eine Extraktionsregel angegeben werden. Dies geschieht durch eine Typdeklaration der Extraktionsvariable im Abbildungs-Block. Die genaue Grammatik steht im Anhang [w4f].

W4F ist in der Lage, NSL auf die Standarddatentypen von Java (`string`, `int`, `float`) und deren Array-Erweiterungen abzubilden. Zur Veranschaulichung ein Beispiel, in dem die Extraktionsvariable `pointers` auf ein zweidimensionales String-Array abgebildet wird. Sie enthält dann den Verweisnamen und die URL aller Anker in der Reihenfolge des Auftretens im HTML-Dokument.

```
String[][] pointers; // mapping layer
pointers= html.body->a[*](.txt # .getAttr(href));
```

Zur Abbildung komplexer NSL-Strukturen muß der Nutzer eigene Java-Objekte (auch die Array-Erweiterungen) angeben, deren Konstruktor NSL konsumieren kann. In der Konstruktorroutine ist dann die Konvertierung der NSL-Struktur in Attribute des Java-Objektes zu programmieren. Diese Form der Abbildung ist nicht deklarativ, da keine Abbildungsregeln mehr spezifiziert werden sondern nur noch ein Interface. Nachfolgend ein Beispiel, in dem die Extraktionsvariable `pointers` auf eine Liste von `Pointer`-Objekten abgebildet wird:

```
Pointer[] pointers; // mapping layer
pointers= html.body->a[*](.txt # .getAttr(href));
// Java-Klasse Pointer
public class Pointer
{
  String name, url;
  public Pointer(NestedStringList in)
  {
    nsl=(NSL_List) in;
    name=((NSL_String) nsl.elementAt(0)).toString();
    url=((NSL_String) nsl.elementAt(1)).toString();
```

```
    }
}
```

W4F stellt für die NSL-Struktur die Klassen `NSL_List` und `NSL_String` zur Verfügung, über die der Zugriff auf die NSL-Elemente abläuft.

Der Abbildungsblock im Wrapper-Quelltext beginnt mit dem Token `SCHEMA`. Danach folgen die Typdeklarationen der Extraktionsvariablen aus den Extraktionsregeln.

2.3.2 JEDI[5]

JEDI (**J**ava based **E**xtraction and **D**issemination of **I**nformation) stammt von der GMD Darmstadt und ist ein Werkzeug zur Erstellung von Wrappern für semistrukturierte Datenquellen. Besonders ausgerichtet ist es aber auf WWW-Datenquellen, die HTML-Dokumente liefern. Es wurde in Java implementiert und bietet eine eigene objektorientierte Skriptsprache zur Programmierung der Wrapper. Dabei teilt JEDI den Wrapper grob in zwei Schichten ein, nämlich in die *Anpassungs-* und *Vermittlungsschicht*. In der Anpassungsschicht erfolgt das Retrieval der HTML-Dokumente von den WWW-Datenquellen, die Extraktion der Daten von Interesse und das Abbilden auf eine Zieldatenstruktur. Dabei gibt es keine klare Trennung der drei Aufgaben im Wrapper-Quelltext. Die Vermittlungsschicht dient als Schnittstelle zum Nutzer. In ihr können Sichten auf die Zieldatenstruktur mittels einer einfachen an SQL angelehnten Anfragesprache definiert werden. Weiterhin gehören Hilfsmittel zur Präsentation der Daten hinzu. In Abbildung 2.4 auf der nächsten Seite ist die JEDI-Architektur dargestellt.

2.3.2.1 Die Anpassungsschicht

Das Retrieval der HTML-Dokumente erfolgt über das HTTP-Protokoll, dabei wird die GET oder POST-Methode benutzt. Es können mehrere Quellen parallel abgefragt werden, jedes Ergebnis-Dokument wird aber für sich gespeichert.

JEDI unterscheidet zwischen *generischen Wrappern* und *spezifischen Wrappern*, die Daten heterogener Datenquellen in eine einheitliches Objektmodell transformieren.

Generische Wrapper sind unabhängig von der Datenstruktur einer konkreten Quelle definierbar, da es ein einheitliches Datenmodell gibt. Das gilt zum Beispiel für das Relationenmodell bei relationalen Datenbank-Management-Systemen oder bei

[5] Die Informationen zu JEDI stammen aus [HFAN98] und [jed].

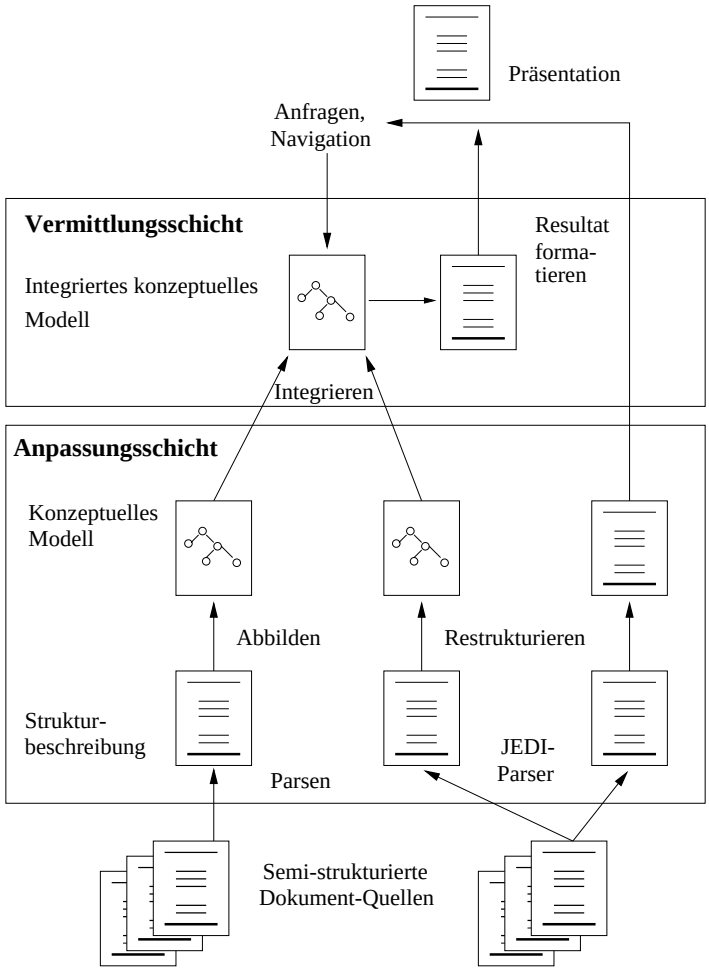

Abbildung 2.4: Die JEDI-Architektur

definierten Austauschformaten für Dokumente wie HTML oder XML. JEDI bietet für HTML-Dokumente mit DOM (**D**ocument **O**bject **M**odel) so eine abstrakte Repräsentation, die mit dem HTML-Analysebaum aus W4F vergleichbar ist. Diese Wrapper können durch Erweiterung des JEDI-Objektmodells mit den entsprechenden Bibliotheken realisiert werden. Mit den Konstrukten der Skriptsprache erfolgt dann der Zugriff auf die Objekte und die Extraktion der gewünschten Daten. Diese werden in einer definierten Zieldatenstruktur, die durch eine Menge von Klassen beschrieben wird, abgespeichert.

Spezifische Wrapper arbeiten auf Quellen, deren Daten semi-strukturiert oder in propitären Formaten vorliegen. Dazu erfolgt das Reverse-Engineering, also das Ermitteln einer expliziten Datenstruktur aus der im Dokument implizit enthaltenen Struktur. Zur Extraktion der Daten werden Regeln spezifiziert, die Struktur und Stellung der gewünschten Datenfragmente im Dokument beschreiben. Die Extraktionsregeln basieren auf mehrdeutigen kontextfreien attributierten Grammatiken, die durch einen robusten fehlertoleranten Parser ausgewertet werden. Die Auflösung der Mehrdeutigkeiten erfolgt dabei durch das Ranking der möglichen Lösungen nach ihrer Spezifität. Die genaue Beschreibung der Analysestrategie ist in [HFAN98] nachzulesen. Die Attributierung der Grammatik geschieht durch Zuweisung von Variablen zu Produktionen der Extraktionsregeln und durch Zuweisung von Programmblöcken zu den Extraktionsregeln. Durch die Angabe von Bedingungen können semantische Prüfungen über syntaktisch gefundenen Daten ausgelöst werden. Die Abbildung der Daten auf die Zieldatenstruktur geschieht entweder im angegebenen Programmblock der Extraktionsregel oder später im Programm.
Es ist möglich, eine aufgestellte Grammatik in einer eigenen Bibliotheksdatei zu speichern, die dann im Wrapper-Quelltext importiert werden kann. Dadurch wird eine gewisse Modularität und damit Wiederverwendbarkeit in anderen Wrappern erreicht.

Um einen Überblick über die Handhabung von JEDI zu geben, folgen nun einige Programmbeispiele für einen generischen und spezifischen Wrapper. Diese stammen aus einem mitgelieferten Beispiel-Wrapper des JEDI-Programmpaketes, der die Abfrage von mehreren Online-Shops zu einem bestimmten Produkt demonstriert. Die Angabe der Zieldatenstruktur erfolgt über ein oder mehrere Klassendefinitionen, die in JEDI als konzeptuelles Modell bezeichnet werden. JEDI benutzt ein schwach getyptes generisches Objektmodell, Objekttypen stehen für mögliche Strukturen, können aber auch andere Strukturen aufnehmen.

```
// Zieldatenstruktur (konzeptuelles Modell)
concept ProductInfo is attributes
  vendor, price, currency, code, description;
end
```

Das Abfragen der WWW-Datenquelle wird nachstehend gezeigt.

```
p1=PrefetchStream(URL("http://www.widget.co.uk/plist.htm"));
```

Dann wird beim generischen Wrapper die HTML-Seite in die abstrakte Repräsentation überführt, hier in die DOM-Darstellung.

```
doc = DOM.fromHTML(p1);
```

Jetzt startet die Extraktion der Daten und die Abbildung auf die Zieldatenstruktur. Im Beispiel werden alle Tabellenzeilen des HTML-Dokumentes selektiert, einzeln ausgewertet und bei Erfolg in der Liste `res1` mit Objekten vom Typ `ProductInfo` als Elemente gespeichert.

```
rows = doc("+/tr");
res1=[];
forall r in rows where r[0]["font"] do
  i=ProductInfo();
  i.vendor="<a href='http://www.widget.co.uk/plist.htm'>
            Widget Software</a>";
  i.code=r[0].text();
  i.description=r[1].text();
  i.currency="$";
  i.price=double(r[3].text()(1)) * 2.93 / 1.73;
  res1.append(i);
end
```

Beim spezifischen Wrapper wird das abgefragte HTML-Dokument zum Extrahieren der Daten an die Startregel der Grammatik übergeben. Diese liefert die Liste `res2` mit Objekten vom Typ `ProductInfo` als Elemente.

```
p2=PrefetchStream(URL("http://www.mplanet.com/cgi/Web_store/web_store.cgi
                       ?page=psion.html&cart_id=2726135.4533"));
planet_com_prices.parse(p2);

rule planet_com_prices :res2 is
  .*
  (res2 += PlanetComProductInfo())+
end
rule PlanetComProductInfo : i is
  ^'<B>' code=('MP' .*) ^'</B>'
  description=@'<'
  @'$' price=[0-9.]+
  .*
  do
```

```
    i=ProductInfo();
    i.vendor="<A HREF=http://www.mplanet.com/cgi/Web_store/web_store.cgi
              ?page=psion.html&cart_id=2726135.4533>MobilePlanet</A>";
    i.code=code;
    i.description=description;
    i.currency="$";
    i.price=double(price);
  end
end
```

2.3.2.2 Die Vermittlungsschicht

In der Vermittlungsschicht werden die einzelnen konzeptuellen Modelle der WWW-Datenquellen zu einem gemeinsamen konzeptuellen Modell integriert. Dieses hat dann ebenfalls die Form eines Objektnetzes. Für Anfragen auf den Objekten stehen Konstrukte der Skriptsprache oder eine einfache SQL-ähnliche Anfragesprache, die in die Skriptsprache eingebettet ist, zur Verfügung.

Die Integration der beiden Objektlisten aus dem vorherigen Beispiel geschieht durch Konkatenation zu einer neuen Liste.

```
list=[];
list.addAll(res1);
list.addAll(res2);
```

Dabei müssen die Elemente des Listenobjektes `list` nicht vom selben Objekttyp sein. Nun folgt ein Beispiel für eine Anfrage nach bestimmten Elementen auf dieser Zieldatenstruktur `list`.

```
result = select x from x in list
         where x isa ProductInfo
               and x.description.upper contains "SERIE";
```

Das Ergebnis der Anfrage ist die Liste `result` mit Objekten vom Typ `Product Info` als Elemente und kann durch die Skriptsprache weiter verarbeitet werden, z. B. Ausgabe einer HTML-Seite mit einer Liste von den gefundenen Produkten.

2.3.3 WebL[6]

WebL wurde speziell für das Retrieval und Verarbeiten von Dokumenten des WWW entwickelt. Es stammt vom Forschungszentrum der Firma Compaq Sy-

[6] Die Informationen zu WebL stammen aus [Mar99].

stems. WebL wurde in Java implementiert und stellt eine eigene imperative Skriptsprache mit speziellen Funktionen für das Informations-Retrieval über HTML-Dokumente dar. Ein WebL-Wrapper besitzt keine klare Schichtentrennung zwischen den Aufgaben Retrieval, Extraktion und Abbildung auf die Zieldatenstruktur. Er ist ein Programm, daß mit einer der zu lösenden Aufgabe angepaßten hochspezialisierten Programmiersprache implementiert wurde. Somit ist so ein Wrapper dem klassischen Design, siehe Abschnitt 2.2.1, zuzuordnen. Trotzdem unterscheidet sich das WebL-Konzept von dem einer klassischen Programmiersprache, da sich einige der in Abschnitt 2.2.2 genannten Design-Richtlinien wiederfinden lassen.

Wenn auch keine modulare Trennung zwischen Retrieval und Extraktion im Programm-Code existiert, so gibt es doch eine logische Trennung. Über die sogenannten *Service Combinators* erfolgt das Abfragen der WWW-Datenquellen mit Methoden des HTTP-Protokolls. Dabei bieten die Service Combinators die Möglichkeit, die Semantik des Retrievals genau zu beschreiben. Das bedeutet für den Nutzer mehr Flexibilät bei der Behandlung von für das WWW typischen Problemen wie schlechte Datendurchsatzraten und Verbindungs- oder Serverausfälle. Die Operationen zur Datenextraktion arbeiten aber nicht auf der Darstellung des HTML-Dokumentes als eine große Zeichenkette, sondern auch hier erfolgt zuerst eine Überführung in eine abstrakte Repräsentation des Dokumentes. Dazu dient die *Markup Algebra*, die ein HTML-Dokument mittels der Typ-Konzepte *tag*, *piece* und *piece set* abstrakt abbildet und darauf Operationen für Datenextraktion und Datenmanipulation definiert.

Eine *Marke* (tag) kennzeichnet ein HTML-Tag oder ein Textsegment des HTML-Dokumentes. Es gibt auch leere Marken "<>", die intern von WebL als Platzmarkierungen bei der Ausführung von Operationen eingefügt werden. Eine Marke ist atomar. In Abbildung 2.5 ist zur Veranschaulichung ein Beispiel angegeben.

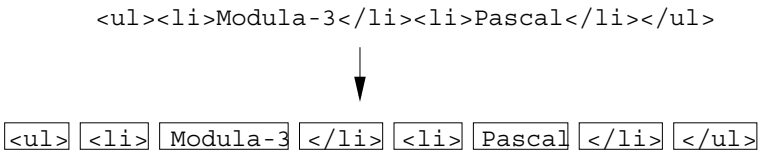

Abbildung 2.5: Abbildung von HTML auf Marken

Ein *Stück* (piece) besteht aus einem zusammenhängenden Bereich einer HTML-Seite, der von einer Start- und einer Endmarke eingegrenzt ist. Dabei gehören diese beiden Marken zum Stück dazu. Ein Stück ist also eine Liste von Marken. Gewöhnlich werden geschlossene HTML-Tags als Stücke abgebildet und mit den Namen der Startmarke benannt. In Abbildung 2.6 auf der nächsten Seite ist ein Beispiel zu sehen.

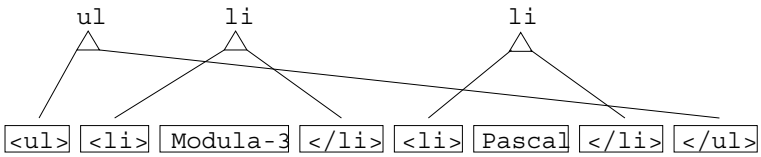

Abbildung 2.6: Bildung von Stücken

Eine *Stückliste* (piece set) ist eine Liste von unterschiedlichen Stücken, die nach der Reihenfolge des Auftretens der Stücke im HTML-Dokument geordnet ist. Stücklisten sind das Ergebnis vieler Operationen der Markup Algebra.

Das nachstehende Beispiel soll einen Eindruck über die Handhabung von WebL liefern. Es gibt den HTML-Quelltext aller Bild-Elemente der HTML-Seite aus. Dabei werden alle "`img`"-Stücke ermittelt und in der Stückliste `images` zusammengefaßt.

```
var P= GetURL("http://www.nowhere.com");
var images= Elem(P, "img");
every image in images do
  PrintLn(image.src)
end
```

Die Skriptsprache bietet nicht nur Operationen zur Datenextraktion sondern auch spezielle Operationen zur Modifikation einer HTML-Seite. z. B. Einfüge-, Ersetzungs- und Löschoperationen. Eine genaue Beschreibung der Skriptsprache ist in [Mar99] zu finden.

2.3.4 Abschließende Betrachtungen zu W4F, JEDI und WebL

Die Wrapper-Toolkits besitzen verschiedene Architekturen und Konzepte aber auch Gemeinsamkeiten. Im folgenden werden die Eigenschaften der drei Toolkits kurz zusammengefaßt und in Tabelle 2.3 gegenübergestellt.

- W4F hat eine klar nach Aufgaben modularisierte Schichtenarchitektur, deren Schichten voneinander unabhängig sind. Das erleichtert Änderungen am Wrapper und vorhandene Lösungen sind leichter wiederverwendbar. Die Verwendung der kleinen (aber mächtigen) deklarativen Spezifikationssprache HEL zur Datenextraktion sichert Verständlichkeit, Übersichtlichkeit und Wiederverwendbarkeit der damit gebildeten Extraktionsregeln. Die gewünschte Unabhängigkeit von einer konkret verwendeten Implementationssprache für den Wrapper bleibt gewahrt. HEL bietet Konstrukte, um auch feinere Strukturen als die HTML-Struktur für die Datenextraktion zu

benutzen.
Ein W4F-Wrapper kann nicht mehrere WWW-Datenquellen gleichzeitig abfragen oder Daten aus mehreren Datenquellen extrahieren und zu einer Datenstruktur zusammenfassen. Sollen mehrere W4F-Wrapper zusammenarbeiten, werden Kontroll- und Datenflußkomponenten benötigt, die eine Integration vornehmen. Dazu dient in der Regel als Integrationsschicht Java. Der W4F-Compiler erzeugt aus den Wrapper-Spezifikationen Java-Klassen, die das Java-Programm einbindet. Dessen Programmcode beschreibt dann die Ablaufsteuerung der Wrapper-Ausführung, den Datenfluß zwischen den Wrappern und die Integration der extrahierten Daten in eine gemeinsame Zieldatenstruktur.
Die vom Toolkit angebotenen Unterstützungswerkzeuge helfen vor allem dem Einsteiger, sich schneller einzuarbeiten. Die gute Dokumentation des W4F-Toolkits ist ebenfalls eine große Hilfe.

- JEDI benutzt eine eigene objektorientierte Skriptsprache zur Wrapper-Erstellung für Datenquellen. Es gibt keine klare Trennung zwischen den einzelnen Aufgaben Retrieval, Extraktion und Abbildung. Allerdings kann die Zieldatenstruktur durch Objektdefinitionen beschrieben werden. JEDI bietet zwei Wrapper-Typen an, generische und spezifische Wrapper. Ein generischer Wrapper benutzt ein existierendes Datenmodell zur abstrakten Repräsentation von Daten einer Datenquelle, während ein spezifischer Wrapper eine spezielle Datenstruktur für eine Datenquelle über eine Grammtik definiert. Beim spezifischen Wrapper existiert eine gewisse Modularisierung, da die Extraktion durch eine Grammatik beschrieben wird.
  Ein JEDI-Wrapper kann mehrere Datenquellen parallel abfragen und extrahieren. Dabei kann JEDI auch feinere Datenstrukturen als HTML ausnutzen. Die Zieldatenstrukturen der einzelnen Datenquellen können mit Konstrukten der Skriptsprache zu einer Zieldatenstruktur integriert werden, die ein Objektnetz darstellt. Darauf kann mit einer einfachen SQL-ähnlichen Anfragesprache, welche in der Skriptsprache enthalten ist, zugegriffen werden.
  Die Skriptsprache bietet Möglichkeiten, um Java-Klassen innerhalb eines JEDI-Wrappers zu benutzen. Umgekehrt kann man auf JEDI-Klassen in einem Java-Programm zugreifen. Die Dokumentation dazu ziemlich ist ziemlich dürftig, man sollte mit einigen Einschränkungen bei der Integration von Java in JEDI bzw. der Integrierbarkeit von JEDI in Java rechnen.

- WebL bietet eine auf das Retrieval und Verarbeiten von HTML-Dokumenten spezialisierte Skriptsprache, die auf einem internen Datenmodell des HTML-Dokumentes arbeitet. Mit WebL geschriebene Wrapper sind ansonsten klas-

sische Programmiersprachen-Wrapper, das Retrieval, die Extraktion und die Abbildung der Daten sind nicht modular.
Es können mehrere WWW-Datenquellen parallel abfragt und zu einem Ergebnis integriert werden. Die Skriptsprache bietet Konstrukte, um auch feinere Datenstrukturen als die HTML-Struktur des Dokumentes auszuwerten. WebL-Programme können auf Java-Klassen zugreifen, umgekehrt aber ist WebL nicht in Java integrierbar. Wie auch bei JEDI gilt, daß mit Einschränkungen zu rechnen ist, wenn unterschiedliche Programmiersprachen integriert werden.

	W4F	**JEDI**	**WebL**
implementiert in	Java	Java	Java
Sprachkonzept	Regeln	Skriptsprache	Skriptsprache
Ausführungskonzept	Compiler	Interpreter	Interpreter
Schichtenarchitektur	ja	teilweise	nein
abstrakte HTML-Repräsentation	ja	ja	ja
Anpassung an Datengranularität	ja	ja	ja
Paralleles Abfragen von Datenquellen	nein[a]	ja	ja
Integration mehrerer Datenquellen	nein[a]	ja	ja
Dokumentation - Theorie	ausführlich	ausführlich	ausführlich
Dokumentation - Praxis	gut	lückenhaft	ausführlich
Integrierbarkeit in Java	100%	möglich	nein
Integration von Java-Klassen	ja	ja	ja

[a] Siehe Text zu W4F ab Seite 22.

Tabelle 2.3: Eigenschaften der Wrapper-Toolkits

Zur Erstellung der Wrapper im Beispiel-Szenario *Reiseverbindungen* (siehe Kapitel 5) wurde von mir das W4F-Toolkit verwendet. Das hat mehrere Gründe.
Mir gefiel das Modularisierungskonzept, das einen Wrapper in seine Teilaufgaben Retrieval, Extraktion und Abbildung aufteilt. Dadurch wird die Verwendung kleiner, klar abgegrenzter, deklarativer Spezifikationssprachen für jede Teilaufgabe möglich. Die Extraktionssprache HEL ist bei der Extraktion von Daten unter Ausnutzung der HTML-Struktur sehr leistungsfähig, obwohl sie nur aus wenigen Sprachkonstrukten besteht. HEL bietet eine bessere Verständlichkeit und damit Wiederverwendbarkeit von Extraktionslösungen als prozedurale Extraktionsbeschreibungen in einer Skriptsprache wie bei JEDI und WebL.
Die Integration mehrerer Wrapper erfolgt nicht durch eine eigene neue Skriptsprache, sondern ist Aufgabe eines Java-Programms. Dadurch steht die ganze

Mächtigkeit von Java und externer Java-Bibliotheken, z. B. für die Datenbankanbindung, sofort zur Verfügung. Mögliche Integrationsprobleme, die zwischen Java und den Skriptsprachen von JEDI und WebL auftreten können, entstehen gar nicht erst.

Ein andere Möglichkeit zur Integration von Wrappern ist die Nutzung eines Workflow-Management-Systems (WfMS). Gerade wenn komplexe Kontroll- und Datenflüsse zwischen Wrappern und anderen Applikationen bestehen, bietet sich das an. In Arbeitsabläufen wird der Kontroll- und Datenfluß explizit modelliert und dann in Workflows des WfMS umgesetzt. Dieses Verfahren wurde bei der Realisierung der integrierten Anfragesicht des Beispiel-Szenarios *Reiseverbindungen* verwendet (siehe Kapitel 5).

2.4 Probleme mit WWW-Datenquellen

Bei der Erstellung von Wrappern für WWW-Datenquellen stößt man recht schnell auf Einschränkungen, die ein automatisches Abfragen von Daten erschweren oder verhindern. Mit dem Einsatz von Javascript, ActiveX, Java-Applets und anderer diverser Plugins zur Darstellung von Informationen auf WWW-Seiten sind diese Daten für Wrapper-Systeme verloren, da keine klare Seitenbeschreibung mehr existiert. Einzig bei Javascript, daß zur Dynamisierung des sonst statischen HTML-Codes benutzt wird, ist eine Auswertung des mit übermittelten Quellcodes denkbar und somit die Ermittlung der dadurch dargestellten Information. Da diese Skriptsprache aber enorm komplex ist, wird man sich wahrscheinlich auf ausgewählte Konstrukte beschränken müssen.

In der Welt des reinem HTML tauchen ebenfalls Probleme bei der automatisierten Datengewinnung auf. Dazu im folgenden einige Beispiele.

- WWW-Seiten ändern sich relativ häufig im Layout. Außerdem können sie teilweise fehlerhaft in HTML codiert sein. Diesem kann man nur mit robusten und leicht anpassbaren Wrappern begegnen.

- Ein großes Übel ist die Ersetzung von Text durch Bilder, die dann nicht mehr auswertbar sind. Dies geschieht leider sehr häufig, obwohl die Übertragungsgeschwindigkeiten im Internet nur mäßig sind, aber so ein individuellere Gestaltung möglich wird.

- Mit der Einführung von Frames in HTML wird dem Designer der WWW-Seiten bei sinnvollem Einsatz eine bessere Nutzerführung ermöglicht. Eine

aus Frames zusammengesetzte WWW-Seite stellt für den Nutzer eine logisch zusammengehörende Einheit dar, obwohl sie aus mehren Teilseiten mit unterschiedlichen URL-Adressen besteht. Für einen Wrapper erhöht sich enorm der Aufwand, da er um die Daten zu ermitteln, mehrere HTML-Seiten abfragen muß.

- Viele HTML-Dokumente werden dynamisch erzeugt, sind also das Ergebnis der Ausführung eines Programmes. Beim automatischen Ausfüllen eines HTML-Formulares durch einen Wrapper kann eine Ergebnisseite erzeugt werden, auf die der Wrapper nicht vorbereitet ist, da dieses Ergebnis noch nie auftrat.

- Ein Programm kann ein HTML-Formular mit dynamisch generierten Variablennamen erzeugen. Einen Programmierer, der einen Wrapper für so ein HTML-Formular schreiben soll, stellt das vor erhebliche Probleme.

Kapitel 3

Arbeitsvorgangs- und Workflow-Modellierung

Dieses Kapitel beschäftigt sich mit den theoretischen Grundlagen der Arbeitsvorgangs- und Workflow-Modellierung und bildet so die Basis für die weiteren Kapitel 4 und 5. Der Abschnitt 3.1 geht auf die prinzipielle Verfahrensweise bei der Modellierung ein. In Abschnitt 3.2 wird die Modellierung von Arbeitsvorgängen erläutert, der Abschnitt 3.3 beschäftigt sich mit der Abbildung der Arbeitsvorgänge auf Workflows. Dazu werden Modellierungskonzepte beschrieben und untereinander verglichen. Der letzte Abschnitt 3.4 zählt wesentliche Anforderungen an ein Workflow-Management-System auf, die es erfüllen sollte.

3.1 Globale Vorgehensweise

Die Workflow-Modellierung hat zum Ziel, eine integrierte Softwarelösung auf Basis eines Workflow-Management-Systems (WfMS) zu schaffen, welche zur Unterstützung von Geschäftsprozessen in Unternehmen dient.

Geschäftsprozesse [JBS97, S. 7] sind größere Einheiten des Geschehens in einem Unternehmen, die einen wesentlichen Beitrag zum Geschäftserfolg leisten, ein klar umrissenes Thema besitzen und ein Aktionsprogramm zur Zielerreichung verwirklichen. Sie legen die Aufgaben eines Unternehmens fest. Die genauere Modellierung des Aktionsprogrammes eines Geschäftsprozesses findet in der Ablauf- und Aufbauorganisation statt. Die Ablauforganisation besteht aus der Arbeitsanalyse, die einzelne Arbeitsschritte identifiziert, und der Arbeitssynthese, die Arbeitsschritte zu Arbeitsvorgängen zusammenfaßt. Diese Arbeitsvorgänge stellen die möglichen Workflows dar, falls die integrierte Softwarelösung mittels

einem Workflow-Management-System erstellt wird. In der Aufbauorganisation wird eine Aufgabenstruktur aus den Arbeitsschritten ermittelt, die als Grundlage für die Bildung von Organisationseinheiten des Unternehmens dient. Nach der Präzisierung und Vervollständigung der Arbeitsabläufe und der Wahl eines konkreten Workflow-Management-Systems erfolgt die Spezifikation in einer konkreten Workflow-Sprache. Daran schließt sich die Implementierungs- und Testphase an, und zuletzt erfolgt der praktische Einsatz und die mögliche Überarbeitung, so daß der gesamte Prozeß mehrfach durchlaufen werden kann. Das Vorgehensmodell ist in Abbildung 3.1 dargestellt. Die Arbeitsvorgangsmodellierung ist

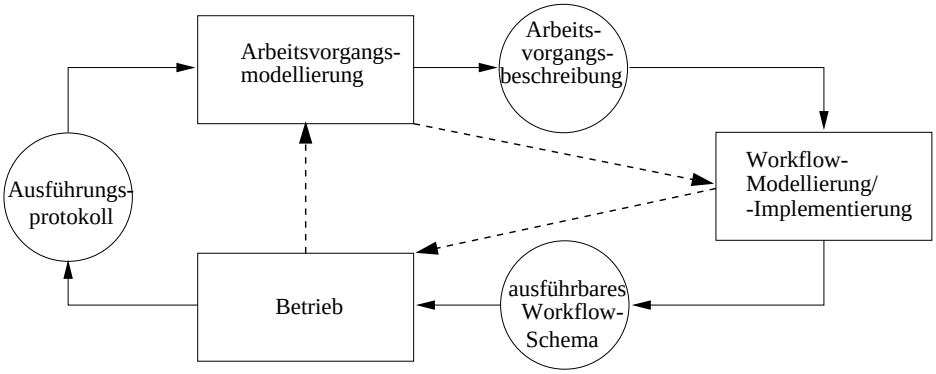

Abbildung 3.1: sequentielles Vorgehensmodell

hier eine selbständige Phase, deren Ergebnisse die Grundlage für die Workflow-Modellierung bilden. Man nennt dies auch sequentiellen Vorgehensansatz. Weitere Ansätze sind in [JBS97, S. 146] zu finden.

3.2 Modellierung von Arbeitsvorgängen

In diesem Abschnitt werden die Grundlagen der Arbeitsvorgangsmodellierung beschrieben. Neben der Klärung von Begriffen wird die Struktur und Zusammensetzung von Arbeitsvorgängen erläutert. Abschließend erfolgt eine allgemeine Betrachtung von Diagrammsprachen als Modellierungswerkzeuge und die Vorstellung einer konkreten Diagrammsprache.

3.2.1 Sichten auf Arbeitsabläufe

Zur Unterscheidung der Begriffe Arbeitsvorgang und Arbeitsablauf müssen diese erklärt werden. Ein *Arbeitsvorgang* ist das als Einheit betrachtete Arbeitsverhalten der Akteure, während ein *Arbeitsablauf* eine Beschreibung der Ablaufstruktur

eines Arbeitsvorganges ist, also ein ontologisches Modell. Man kann auch sagen, es interessiert weniger das Ziel eines Arbeitsvorganges als die Art und Weise der Zielerreichung [JBS97, S. 38/39].

Ein Arbeitsablauf setzt sich aus dem Zusammenspiel mehrerer unterschiedlicher Komponenten zusammen, welche durch die Analyse des Arbeitsvorganges ermittelt werden. Folgende Sichten auf Arbeitsabläufe [JBS97, S. 48/49] sind wesentlich und werden zur späteren Spezifikation der zugehörigen Workflows benötigt:

- Die *Extension* beschreibt, welche Arbeitsschritte (Aktivitäten) in welcher Reihenfolge unter Berücksichtigung gegebener Abhängigkeiten durchgeführt werden.
- In der *Intension* sind die Ziele einer Aktivität und des gesamten Arbeitsablaufes beschrieben. Dies ermöglicht, Aussagen über die Zielerreichung von Aktivitäten zu gewinnen.
- In der *Handhabung* wird kurz festgelegt, welche Operationen (z. B. Starten, Abbrechen, Anhalten, Fortsetzen, ...) für einen Arbeitsvorgang eines Arbeitsablaufes möglich sein sollen. Die Umsetzung ist eine konkrete Aufgabe der Workflow-Modellierung.
- In der *Koordination* sind Einschränkungen in der Mittelnutzung durch Aktivitäten festgelegt, um Konflikte bei der Ressourcennutzung zu vermeiden.
- Die *Operation* bestimmt, wer welche Aktivität mit welchen Mitteln durchführen darf.
- Die *Organisation* beschreibt die Mitgliedschaften und Kooperationsbeziehungen der Akteure zu den Organisationseinheiten und der Organisationseinheiten untereinander.

Das Ergebnis der Arbeitsvorgangsmodellierung, die Arbeitsabläufe, sind in einer semi-formalen Sprache beschrieben, wobei zur Darstellung der Extension oft Diagrammsprachen aufgrund der besseren Anschaulichkeit, auch für den Laien, eingesetzt werden.

3.2.2 Struktur von Arbeitsvorgängen

Die Wahl bzw. Entwicklung einer Workflow-Sprache ist nicht unabhängig von den zu modellierenden Arbeitsvorgängen. Sie muß mächtig genug sein, um diese

semantisch richtig darstellen zu können. Arbeitsvorgänge kann man aufgrund ihrer Struktur und des Arbeitsablaufes in mehrere Arten einteilen[1] :

- *Arbeitsvorgänge mit fester Ausführungsreihenfolge der Teilaufgaben* sind gut strukturiert. Es werden nur einfache Kontrollflußkonstrukte sowie Schleifen zur wiederholten Ausführung benötigt, Aktivitäten werden sequentiell, alternativ oder nebenläufig ausgeführt.

- *Arbeitsvorgänge mit bekannten Teilaufgaben, aber Freiheiten in ihrer Abfolge* besitzen keine fest vorgegebene Ausführungsreihenfolge, sondern es existieren mehrere richtige Varianten. Es muß Kontrollflußkonstrukte geben, die das realisieren können.

- *Arbeitsvorgänge mit bekannten Teilaufgaben in situationsabhängiger Abfolge* besitzen keine Struktur der Ausführungsreihenfolge mehr. Alle theoretisch zulässigen Verbindungen zwischen den Aktivitäten könnten situationsbedingt auftreten. Eine explizite Ablaufbeschreibung ist deswegen nicht brauchbar. Die Workflow-Sprache muß Konstrukte bieten, die eine implizite Definition der Ausführungsreihenfolge der Aktivitäten erlaubt. Das ist erreichbar durch Angabe von Vorbedingungen zu jeder Aktivität, bei deren Erfüllung es zur Aktivtitätsausführung in Abhängigkeit von der Situation zur Laufzeit kommt.

Weiterhin hat die Art und Weise, wie Arbeitsvorgänge von den Entwicklern gesehen werden, Einfluß auf die spätere Workflow-Sprache. Das kann dazu führen, daß nach der Wahl einer Workflow-Sprache nicht mehr alle unterschiedlichen Arbeitsvorgänge modellierbar sind. Häufig vorkommende Sichtweisen auf Arbeitsvorgänge[2] sind nachstehend aufgeführt.

- *Arbeitsvorgänge als strukturierte Abfolge von Teilschritten* benötigen ein vorgangsorientiertes Workflow-Sprachmodell. Der Arbeitsvorgang ist eine strukturierte Abfolge von Teilschritten, deren funktionale Dekompositon nötig ist.

- *Arbeitsvorgänge als migrierende Objekte* fokussieren sich auf die Objekte, die Gegenstand der Ausführung von Arbeitsschritten sind und den Arbeitsvorgang erst ermöglichen. Als Beispiel sind die Umlaufmappen genannt. Workflow-Sprachmodelle, die diesen Ansatz verfolgen, nennt man Objektmigrationsmodelle.

[1] nach [JBS97, S. 90]
[2] nach [JBS97, S.91/92]

- *Arbeitsvorgänge als Menge zustandsbehafteter Objekte* richten ihre Aufmerksamkeit auf Daten des Arbeitsvorganges, die von den Arbeitsschritten verändert werden. Den Daten werden explizit Zustände von den Bearbeitern zugewiesen, die das Fortschreiten der Bearbeitung darstellen. Diese Modelle nennt man Information-Sharing-Modelle aufgrund der gemeinsamen Sicht auf die Daten.

3.2.3 Nutzung von Diagrammsprachen

Mit Diagrammsprachen als Modellierungswerkzeuge von Arbeitsvorgängen und damit den zugehörigen Workflows wird eine höhere Anschaulichkeit und Selbsterklärung erreicht. Dadurch fällt es erheblich leichter, das Wissen der Mitarbeiter aus den zu modellierenden Geschäftsbereichen mit einzubeziehen. Das führt letztendlich zu einer höheren Qualität des Entwurfes, die Diagramme bleiben auch für den Laien nachvollziehbar. Eine geeignete Diagrammsprache erfüllt mindestens die folgenden Anforderungen[3].

- Die Darstellung von elementaren Aktivitäten ist möglich. Zwischen Aktivitäten gibt es zeitliche und logische Abhängigkeiten, welche als Kontrollfluß grafisch darstellbar sind. Hierbei sollten umfangreiche Modellierungskonstrukte durch die Diagrammsprache angeboten werden, da dieser Punkt der Hauptzweck für den Einsatz einer Diagrammsprache ist.

- Die Modellierung des Datenflusses, von organisatorischen und technischen Ressourcen sollte möglich sein, allerdings reicht hier auch eine Attributierung der Aktivitäten oder Kanten.

- Zur Einschränkung der Schemakomplexität ist die Unterstützung von vertikaler Hierarchisierung und horizontaler Unterteilung sehr hilfreich.

Die Modellierung der Kontrollflußlogik kann implizit oder explizit erfolgen. Bei der *expliziten Modellierung* wird das Kontrollflußkonstrukt durch einen eigenen Knoten dargestellt und anschließend durch Angabe von auswertbaren Ablaufbedingungen konkretisiert. Die Hervorhebung des Kontrollflußkonstruktes als eigenen Knoten erhöht die Anschaulichkeit und deckt syntaktische Fehler zwischen nachstehenden Kontrollflußknoten auf. So muß ein XOR-Split wieder von einem XOR-Join zusammengeführt werden, unabhängig von den angegebenen Ablaufbedingungen. Bei der *impliziten Modellierung* wird der Kontrollfluß ausschließlich durch Übergangsbedingungen realisiert. Dadurch ist der Kontrollfluß nicht

[3] nach [JBS97, S.162]

immer sofort ersichtlich, es reduziert sich aber die Knoten- und Kantenzahl erheblich. Ein Beispiel für beide Modellierungsarten zeigt Abbildung 3.2.

Im Abschnitt 3.2.3.1 wird eine Diagrammsprache mit expliziter Kontrollflußlogik, die obige Anforderungen erfüllt, vorgestellt.

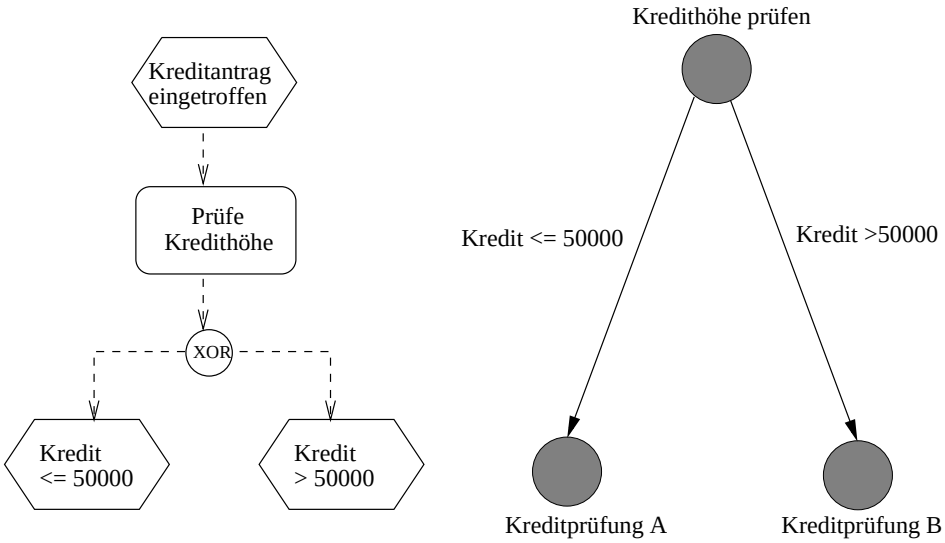

Abbildung 3.2: Explizite und implizite Modellierung der Kontrollflußlogik

3.2.3.1 Ereignisgesteuerte Prozeßketten (EPK)

Ereignisgesteuerte Prozeßketten sind gerichtete, bipartite Graphen mit Ereignissen und Funktionen als Knotentypen[4].

Dabei stellt ein *Ereignis* einen Zustand dar, von dem der weitere Verlauf abhängt. Sie können Funktionen auslösen (Auslöseereignisse) und das Ergebnis von Funktionen repräsentieren (Bereitstellungsereignisse). Jedes Workflow-Schema beginnt mit einem oder mehreren Startereignissen, deren Eintritt zur Instanziierung des Workflow-Schemas führen, und endet mit einem oder mehreren Ereignissen, die am Ende der Ausführung der Workflow-Instanz ausgelöst werden. Das grafische Symbol eines Ereignisses ist ein Sechseck.

Die *Funktionen* stellen die Aktivitäten dar und verarbeiten die Eingabe- und Ausgabedaten. Sie können Zeit benötigen und Ressourcen belegen. Die Ausführung einer Funktion erfolgt durch den Eintritt eines oder mehrerer Ereignisse und hat ein oder mehrere Ereignisse als Ergebnis. Das grafische Symbol einer Funktion

[4]nach [JBS97, S. 167]

ist ein abgerundetes Rechteck. Funktionssymbole können der vertikalen Hierarchisierung dienen, sie sind dann Platzhalter für ein Subworkflow-Schema und werden mit einem Punkt gekennzeichnet. Eine horizontale Unterteilung ist durch die Verwendung von Prozeßwegweisern realisierbar.

Für nichtsequentielle Abläufe gibt es standardmäßig die drei *Verknüpfungsoperatoren* Konjunktion, Adjunktion und Antivalenz. Komplexere Auswahlbedingungen können durch Angabe von Entscheidungstabellen realisiert werden. Dazu dient der ET-Operator. Das grafische Symbol für einen Operator ist ein kleiner Kreis mit dem Symbol für die Verknüpfung darin.

Der *Kontrollfluß* wird mit gerichteten Kanten hervorgehoben, welche Ereignisse, Funktionen und Operatoren verbinden. Eine grafische Darstellung des *Datenflusses* existiert in EPK nicht, für jede Funktion sind aber Eingangs- und Ausgangsdaten angebbar. Den Funktionen können Symbole von organisatorischen und technischen Ressourcen zugeordnet werden, die den Ressourcenbedarf der Funktion kennzeichnen.

Diese Diagrammsprache wird zur Modellierung der Arbeitsabläufe des Beispiel-Szenarios in Kapitel 5 benutzt, da sie sich aufgrund des Ereigniskonzeptes und der expliziten Darstellung des Kontrollflusses gut zur Umsetzung in die verwendete Workflow-Sprache eignet.

3.3 Modellierung von Workflows

Auf die Arbeitsvorgangsmodellierung setzt die Workflow-Modellierung auf. Um Klarheit über die verwendeten Begriffe und deren Zusammenhang untereinander zu schaffen, erfolgt in Abschnitt 3.3.1 eine Einführung. Desweiteren werden in Abschnitt 3.3.2 Modellierungskonzepte für die Funktionsstruktur eines Workflows vorgestellt und anhand von Qualitätskriterien bewertet. In Abschnitt 3.3.3 sind die wesentlichen zu modellierenden Aspekte eines Workflows, die ein Workflow-Metaschema bieten muß, beschrieben. Weiterhin wird dort auf die Überführung von Arbeitsabläufen in Workflow-Schemata eingegangen.

3.3.1 Begriffserklärungen [5]

Es bedarf der Erläuterung einiger Begriffe aus der Workflow-Modellierung, um Mißverständnisse zu vermeiden. Der Zusammenhang zwischen den Begriffen ist in der Abbildung 3.3 auf Seite 35 dargestellt.

[5] nach [JBS97, S. 67ff.]

Ein *Workflow* ist ein Vorgang, dessen Arbeitsschritte durch den Einsatz eines Workflow-Management-Systems gestartet, organisiert, unterstützt, kontrolliert, gesteuert, koordiniert oder ausgeführt werden.

Das *Workflow-Schema-Modell* ist die Idee hinter einem Verfahren (Geschäftsprozeß), das von einem Workflow-Management-System unterstützt werden soll. Es ist oft gebrauchssprachlich formuliert.

Das *Workflow-Sprachmodell* ist die Idee hinter einer Workflow-Sprache. So sind beispielsweise Petri-Netze die Idee hinter einigen Workflow-Sprachen.

Die *Workflow-Sprache* ist eine formale Sprache, die von einem Workflow-Management-System interpretiert werden kann. Mit ihr werden Workflow-Schemata niedergeschrieben. Sie basiert auf einem Workflow-Sprachmodell.

Das *Workflow-Schema* ist eine vollständige, eindeutige, korrekte und widerspruchsfreie Beschreibung des Workflow-Schema-Modells in einer formalen Sprache. Es kann von einem Workflow-Management-System zur Steuerung von Workflows benutzt werden. Daraus ergibt sich zwangsläufig die Festlegung auf eine konkrete Workflow-Sprache.

Die *Workflow-Instanz* ist eine konkrete Ausprägung des Workflow-Schemas, entstanden durch die Interpretation eines Workflow-Management-Systems. Sie enthält alle Daten, die für die Ausführung eines Workflows relevant sind.

Ein *Workflow-Metaschema* gibt alle Möglichkeiten der Workflow-Sprache bei der Erstellung von Workflow-Schemata an. Damit ist es möglich, ein Workflow-Management-System hinsichtlich seiner Modellierungsfähigkeiten zu beurteilen.

3.3.2 Aufbau und Funktionsstruktur von Workflows

Zur Modellierung von Workflows wird hier das vorgangsorientierte Workflow-Sprachmodell gewählt, d. h. man kann eine Funktionsstruktur zum Workflow als Ergebnis der funktionalen Dekomposition des Arbeitsablaufes ableiten. Zur Darstellung von Funktionsstrukturen werden Bausteine und Regeln für das Zusammensetzen zu einer Struktur benötigt. Diese stellt die Workflow-Sprache zur Verfügung. Es kann mehrere unterschiedliche Bausteintypen geben, der Typ eines Bausteins bestimmt die Stellung in der Funktionshierarchie. Begriffe wie Prozeß, Aktivität, Workflow bezeichnen solche Bausteintypen. Wichtig ist die Abgrenzung nach außen und die Darstellung der internen Struktur mit Hilfe dieser Bausteine. Zur Einordnung von Workflow-Management-Systemen, die sich eines solchen Modells bedienen, wurden in [JBS97, S. 107/108] einige Qualitätskriterien festgelegt.

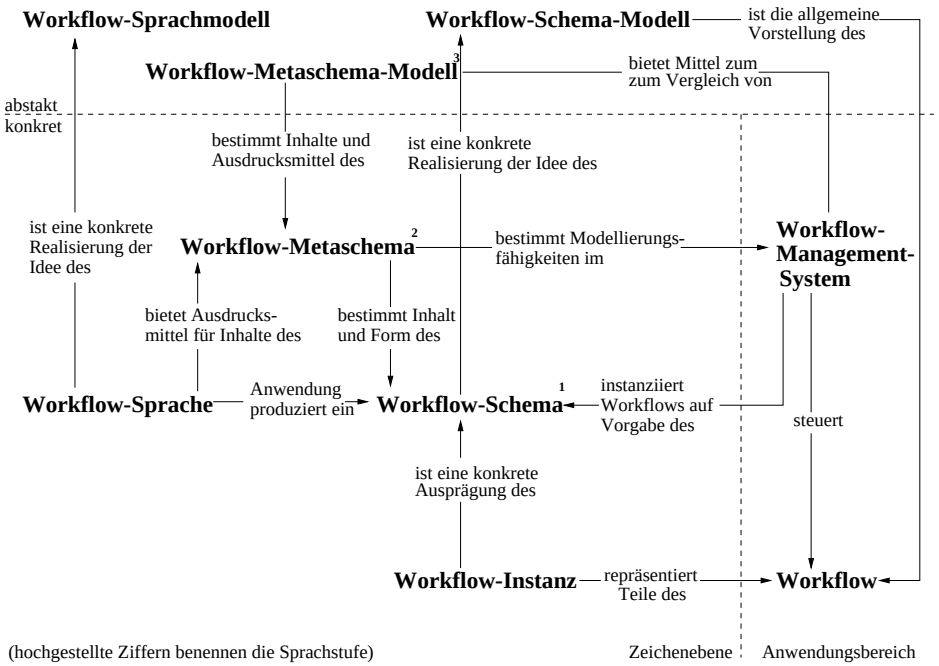

Abbildung 3.3: Begriffe bei der Modellierung von Workflows

- Die *Übersichtlichkeit* fordert, daß auch Workflows zu komplexen Arbeitsvorgängen klar und verständlich dargestellt werden können. Dafür erforderlich ist das Vorhandensein von Konstrukten, die eine Gruppierung von Aktivitäten ermöglichen und die Darstellung auf unterschiedlichen Abstraktionsebenen zulassen. Hier sind nicht unterstützende Designwerkzeuge gemeint, wie es sie bei grafisch orientierten Workflow-Sprachen gibt, sondern die Sprachelemente selbst, also auf konzeptueller Ebene.

- Die *Wiederverwendbarkeit* fordert, daß sich Unterstrukturen von Workflows identifizieren und isolieren lassen, damit bei der Spezifikation anderer Workflows eine Wiederverwendung möglich ist.

- Die *Änderungsfähigkeit* berücksichtigt, daß Arbeitsvorgänge Veränderungen unterworfen sind. So ändert sich die Funktionsstruktur des Workflows, Aktivitäten kommen hinzu, fallen weg oder werden verfeinert. Es sollten nur die betroffenen Unterstrukturen überarbeitet und nicht der gesamte Workflow neu erstellt werden müssen.

Es gibt weitere Qualitätskriterien, aber anhand dieser genannten lassen sich die in Abschnitt 3.3.2.1 vorgestellten Konzepte zur Darstellung von Funktionsstrukturen gut bewerten.

3.3.2.1 Workflow-Sprachmodelle zum Aufbau von Funktionsstrukturen

Der Baum ist eine gut geeignete Darstellungsform für die Funktionsstruktur eines Workflows. Er hat eine Ebenendarstellung, an der man eine Ober- und Unterstruktur erkennen kann. Die Oberstruktur sind in der Regel die Knoten, die Unterstruktur ist die zugeordnete Funktionsstruktur. Im folgenden sollen einige Modelle zum Aufbau von Funktionsstrukuren von existierenden Workflow-Sprachen beschrieben und ihre Vor- und Nachteile aufgezeigt werden (nach [JBS97, S. 109]).

Bei *zwei expliziten Bausteintypen zur Funktionsbeschreibung* gibt es den Bausteintyp „Prozeß“ und den Bausteintyp „Aktivität“. Der *Prozeß* faßt den Workflow als Einheit nach außen zusammen und ist somit die Oberstruktur. Aus den *Aktivitäten*, den einzelnen Teilschritten, wird die Unterstruktur gebildet. Einem Prozeß können beliebig viele Aktivitäten zugeordnet werden, welche alle auf der gleichen Stufe der Funktionshierarchie stehen. Durch die erzwungene Tiefe von 1 des Baumes ist man in der Darstellung äußerst eingeschränkt. Dieses Modell kommt in existierenden Workflow-Management-Systemen nicht vor.

Eine Erweiterung um einen Bausteintyp führt zu *drei expliziten Bausteintypen zur Funktionsbeschreibung*. Die Typen heißen jetzt „Prozedur“ (Oberstruktur), „Aktivität“ und „Aktion“ (Unterstruktur). Eine *Aktion* ist die kleinste Einheit und stellt somit die Blattebene des Strukturbaumes dar. Eine *Aktivität* kapselt mehrere Aktionen und ist selbst unmittelbar der *Prozedur* untergeordnet. Die Übersichtlichkeit ist leicht verbessert, da man Aktionen gruppieren kann, allerdings ist der Baum auf eine Tiefe von 2 beschränkt. Dieses Modell kommt bei dem Workflow-Management-System FlowPath [Bul93] zum Einsatz.

Bei *drei expliziten Bausteintypen mit rekursivem Konstruktionsprinzip* wird zugelassen, daß der Baum eine beliebige Tiefe haben kann. Es gibt die Bausteintypen „Prozeß“ (Oberstruktur), „Block“ (Binnenstruktur) und „Aktivität“ (Unterstruktur). Der *Block* kapselt mehrere *Aktivitäten*, muß aber nicht mehr unmittelbar dem *Prozeß* untergeordnet sein. Ein Block darf jetzt auch dort stehen, wo eine Aktivität erlaubt ist. Somit gibt es nun auch eine vertikale Gliederung der Struktur. Dieses Modell wird bei dem Workflow-Management-System FlowMark [IBM96] und bei der Workflow-Management-Coalition [Wor96] verwendet.

Im Gegensatz zu den bisher vorgestellten Modellen gibt es das Modell der *verwendungsgebundenen Typisierung eines Bausteines*. Es existiert nur ein Bausteintyp „Workflow“, der an allen Stellen der Funktionstruktur stehen darf. Das heißt, ein *Workflow* kann für den gesamten Workflow als Oberstruktur, für eine Gruppierung von Aktivitäten als Binnenstruktur oder für eine einzige elementare Aktivität als Unterstruktur stehen. Der Typ ergibt sich implizit aus der Stellung im Funktionsbaum und wird hier nicht fest vorgegeben. Man spricht dann je nach Verwendung

des Bausteines auch vom kompositen bzw. zusammengesetzten Workflow oder vom elementaren Workflow. Zum Einsatz kommt dieses Modell in den Workflow-Management-Systemen MOBILE [JB96] und WorCOS [SBMW+96].

Zur Verdeutlichung ist die Struktur der Modelle in Abbildung 3.4 veranschaulicht.

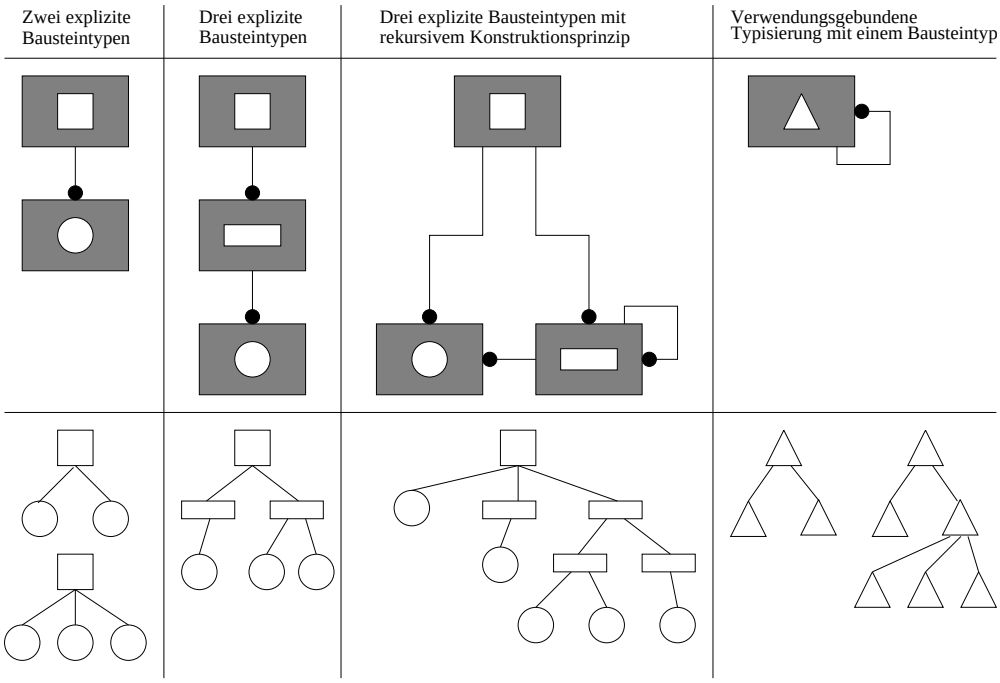

Abbildung 3.4: Workflow-Sprachmodelle zur Darstellung der Funktionsstruktur eines Workflows [JBS97, S. 110]

3.3.2.2 Eignung der Modelle bei Änderung der Funktionsstruktur

Die vorgestellten Modelle erfüllen das wichtige Qualitätskriterium *Änderungsfähigkeit* sehr gut bis schlecht. Dies wird am Beispiel der Verfeinerung der Funktionsstruktur des in Abbildung 3.5 auf der nächsten Seite skizzierten Workflows dargestellt. Dort soll die Aktivität c durch drei neue Aktivitäten d, e und f verfeinert werden.

Bei zwei expliziten Bausteintypen (Abbildung 3.5, Spalte 1) ist dies nicht erlaubt, da keine weiteren Unterstrukturen zugelassen sind. Es wäre möglich, daß die Aktivität c auf einen neuen Workflow S mit der Unterstruktur aus d ,e und f zeigt (Abbildung 3.5, Spalte 2). Allerdings ist der Zusammenhang zwischen den Aktivitäten nicht mehr richtig wiedergegeben. Bei drei expliziten Bausteintypen

(Abbildung 3.5, Spalte 3) wird die Aktivität c durch den Blockbaustein ausgetauscht, dem dann die Unterstruktur zugeordnet wird. Änderungen betreffen nur die zu verfeinernde Stelle der Funktionsstruktur. Ohne rekursives Konstruktionsprinzip ist die Tiefe aber beschränkt, so daß dann dieselben Probleme wie bei zwei expliziten Bausteintypen auftreten. Da bei der verwendungsgebundenen Typisierung (Abbildung 3.5, Spalte 4) ein Bausteintyp für elementare Aktivitäten oder komplexe Substrukturen stehen darf, wird die neue Substruktur einfach an der betroffenen Stelle angehängt. Aufgrund der Einfachheit bei der Änderung der Funktionsstruktur habe ich mich an diesem Modell bei der Entwicklung des Mini-Workflow-Management-Systems orientiert.

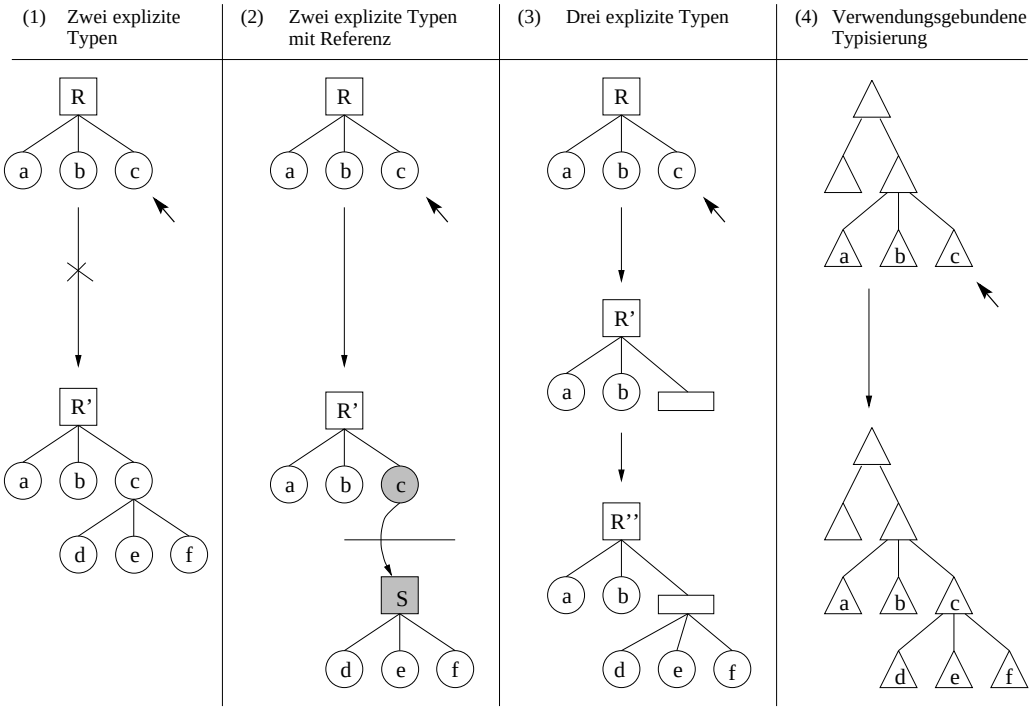

Abbildung 3.5: Verfeinerung der Funktionsstruktur eines Workflows [JBS97, S. 113]

3.3.3 Die Bedeutung des Workflow-Metaschemas

Das Workflow-Metaschema eines Workflow-Management-Systems legt alle zur Verfügung stehenden Konstrukte fest, mit denen Workflow-Schemata spezifiziert werden können. Komplexere Schemata erlauben vermutlich eine größere Ausdrucksvielfalt als einfachere. Ein Workflow-Metaschema sollte folgende wichtige Aspekte [JBS97, S. 122] abdecken:

- Der *Funktionsaspekt* stellt Konstrukte zur Beschreibung der Funktionsstruktur eines Workflow-Schemas zur Verfügung. Die Funktionsstruktur definiert eine hierarchische Aufgabenzerlegung. Weiterhin werden Workflow-Operationen zu dem Workflow-Schema angegeben, so daß nur die benötigten Operationen des Workflow-Metaschemas zur Verfügung stehen.

- Die Konstrukte des *Informationsaspektes* ermöglichen die Modellierung der Datenstruktur und des Datenflusses im Workflow-Schema. Workflows werden zur Laufzeit mit Parametern versorgt und geben Parameter an die Subworkflows weiter. Die Ergebnisse von Subworkflows können an andere Subworkflows übergeben, in lokalen Variablen gespeichert oder als Rückgabeparameter verwendet werden.

- Der *Verhaltensaspekt* dient zur Definition des Kontrollflusses des Workflow-Schemas, indem dessen Subworkflows in eine Ablaufreihenfolge angeordnet werden. Dies geschieht mittels Konstrukten, die Sequenz, Parallelität, bedingte Verzweigungen, Wiederholungen usw. modellieren.

- Der *Organisationsaspekt* umfaßt die Aufbauorganisation des Unternehmens und die Zuordnung von Ausführenden zu den Aktivitäten, ob nun Mensch oder Maschine.

- Im *Operationsaspekt* wird die Einbindung von externen Applikationen in die Workflow-Ausführung geregelt. Dazu müssen Workflow-Applikationen erstellt werden, die die Integration bewerkstelligen und eine einheitliche Aufrufschnittstelle für das Workflow-Management-System bieten.

Wichtig ist, daß sich alle relevanten Konstrukte aus den Arbeitsabläufen mit Konstrukten des Workflow-Metaschemas semantisch richtig darstellen lassen, so daß eine korrekte Abbildung des Arbeitsablaufes im Workflow-Schema entsteht. Sollte ein Konstrukt keine semantische Entsprechung in der Workflow-Sprache besitzen, muß es mit mehreren Konstrukten der Workflow-Sprache nachbildbar sein, ohne seine Semantik zu verändern. Ein Beispiel hierfür ist die Darstellung einer `CASE`-Anweisung mit Hilfe einer geschachtelten `IF-THEN`-Anweisung. Ansonsten ist eine Ergänzung des Workflow-Metaschemas um ein entsprechendes Konstrukt unumgänglich, oder es darf im Quellschema nicht verwendet werden.

Das Abbilden auf semantisch ähnliche Konstrukte, wo neben strukturellen auch inhaltliche Änderungen am Schema notwendig sind, darf nur erfolgen, wenn kleine überschaubare Bereiche betroffen sind. So kann eine Parallelität im Arbeitsablauf durch eine Sequenz im Workflow-Schema dargestellt sein, wenn das Workflow-Metaschema keine Parallelitäten kennt. Man sollte dieses Vorgehen möglichst vermeiden, da sich der Zusammenhang zwischen Arbeitsablaufschema und

Workflow-Schema verliert und die Gefahr von Fehlern bei der Überführung wächst. Weiterhin können Ergebnisse von Tests und Simulationen auf den Arbeitsabläufen verändert werden.

Anfallende Optimierungen und Reengineering-Maßnahmen auf dem Arbeitsablaufschema können bei direkter Überführung der Konstrukte ohne großen Aufwand, eventuell sogar automatisiert, ins Workflow-Schema übertragen werden. Somit wird eine einfache Wartbarkeit garantiert und Überführungsfehler werden im Vorfeld eingeschränkt.

3.4 Anforderungen an ein Workflow-Management-System

Weitere Anforderungen an ein „modernes“ Workflow-Management-System betreffen neben den bereitgestellten Möglichkeiten der Workflow-Modellierung auch den stabilen Betrieb und die Erweiterungsmöglichkeiten des Systems. Die nachstehenden Anforderungen wurden aus [JBS97, S. 219] entnommen. Es werden funktionale und nichtfunktionale Anforderungen unterschieden. Funktionale Anforderungen an ein Softwaresystem erklären die benötigten Funktionen und deren Schnittstellen mit Eingabe- und Ausgabeparametern. Die nichtfunktionalen Anforderungen stellen Bedingungen an die Funktionsausführung wie Einhaltung bestimmter Antwortzeiten oder Verfügbarkeitskriterien.

3.4.1 Funktionale Anforderungen

Diese werden vorrangig durch die Workflow-Sprache bestimmt. Ein Workflow-Management-System muß sein Workflow-Metaschema korrekt implementieren. Dabei gelten die Anforderungen an ein gutes Workflow-Metaschema auch für das Workflow-Management-System. Es muß in der Lage sein,

- Workflows bestehend aus Subworkflows und Workflow-Applikationen zu verwalten (Funktionsaspekt),
- die Einhaltung des Kontrollflusses zu überwachen und den Benutzer zu lenken (Verhaltensaspekt),
- Kontroll- und Produktionsdaten eines Workflows zu verwalten sowie den Datenfluß zwischen verschiedenen Workflows und Workflow-Applikationen herzustellen (Informationsaspekt),

- Workflows an Akteure zuzuweisen und somit die Organisationsstruktur des Unternehmens zu verwalten (Organisationsaspekt), sowie
- externe Programme aus den Workflow-Applikationen aufzurufen (Operationsaspekt).

Dabei ist zu bemerken, daß nur geringe Teile der Produktionsdaten eines Workflows (und somit Arbeitsvorganges) von dem Workflow-Management-System kontrolliert werden. Der Hauptteil wird durch externe Programme verwaltet, die zur Ausführung des Arbeitsschrittes benutzt werden.

Zu den funktionalen Anforderungen gehören auch die Benutzerschnittstellen, die in die Anwender-, Entwickler- und Administratorschnittstelle einteilbar sind. Der *Anwender* benutzt das Workflow-Management-System zur Erledigung seiner anfallenden Arbeit. Dabei benötigt er die Arbeitsliste, in der zugeteilte Aufgaben, also Workflow-Instanzen, mit den zugehörigen Informationen wie Name, Prioritäten, Eingangszeiten usw. verzeichnet sind. Außerdem braucht er eine Liste mit Arbeitsvorgängen, also Workflows, die er starten darf, und er kann möglicherweise Kontrollfunktionen wie Abbrechen, Weiterleiten, Beenden von Workflows ausüben. Der *Entwickler* erstellt Workflow-Schemata. Dafür benötigt er eine Komponente zum Einbringen der Schemadefinitionen, eine Analysekomponente zur Fehlerüberprüfung und eine Verwaltungskomponente für die Workflow-Schemata. Der *Administrator* braucht eine Komponente für die Benutzerverwaltung und ein Konfigurationswerkzeug für das Workflow-Management-System. Weiterhin muß er in die Workflow-Bearbeitung eingreifen können und Zugriff auf Monitoring- und Statistikfunktionen haben.

3.4.1.1 Implementierungsarchitekturen

Eine Implementierungsarchitektur für ein Workflow-Management-System ist in Abbildung 3.6 auf der nächsten Seite dargestellt. Alle funktionalen Aspekte werden durch eine operationale Sprache abgedeckt, d. h. es existiert eine operationale Ausführungssemantik. Ein Interpretermodul verarbeitet das in der operationalen Sprache vorliegende Workflow-Schema einer Workflow-Instanz und führt die zugehörigen Operationen aus. Dieses Modul ist so auch die Koordinationskomponente zwischen den Aspekten. Durch Erweiterung der operationalen Sprache können weitere Aspekte hinzugefügt werden, z. B. für Fehlerbehandlung. Die Hilfsmodule dienen dem Aufruf von externen Applikationen und der Anbindung von Werkzeugen.

Diese Architektur habe ich für die Realisierung des Mini-Workflow-Management-Systems gewählt. Weitere Implementierungsarchitekturen für die funktionalen Aspekte sind in [JBS97, S. 233] nachlesbar.

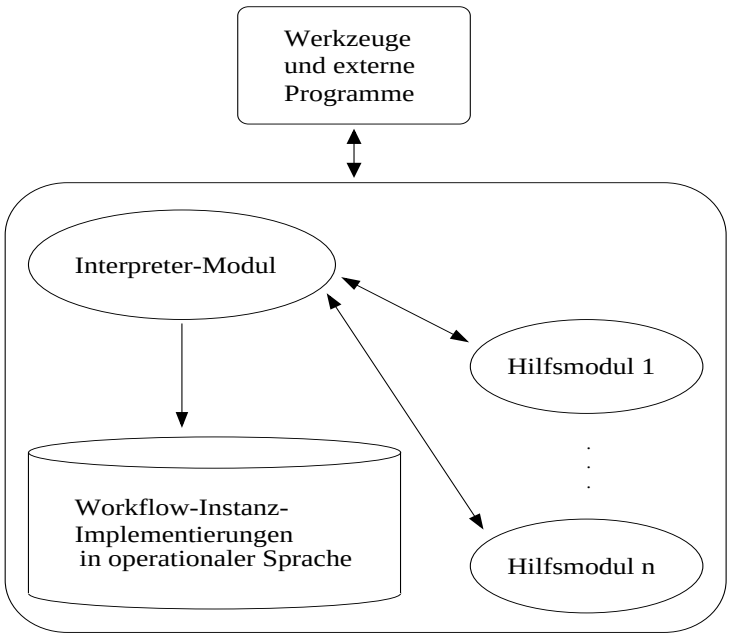

Abbildung 3.6: Implementierungsarchitektur mit Workflow-Interpreter [JBS97, S. 240]

3.4.2 Nichtfunktionale Anforderungen

Eine wichtige Forderung an ein Workflow-Management-System ist Offenheit in Bezug auf Erweiterbarkeit der Funktionalität und deren Optimierung, Integrierbarkeit in bestehende Infrastrukturen, Integration von externen Applikationen, Plattformunabhängigkeit aufgrund heterogener Netzwerke sowie Skalier- und Verteilbarkeit für Anpassungsfähigkeit gegenüber Veränderungen in der Umgebung. Abbildung 3.7 auf der nächsten Seite zeigt eine Schichtenarchitektur, die diese Anforderungen erfüllt. Die unterste Schicht bilden das Betriebsystem, das Datenbanksystem und rudimentäre Kommunikationsdienste. Darüber liegt die Client/Server-Kommunikationsschicht, die für Transparenz bezüglich Verteilung und Heterogenität sorgt. Die anschließende Verfügbarkeitsschicht verwendet Replikationsmechanismen, um Komponentenausfälle transparent zu halten. Die Skalierungsschicht kümmert sich um Lastverteilung durch Partitionierung von Workflow-Daten und durch Wahl einer der Situation angepaßten Scheduling-Strategie für die Workflow-Instanzen. Die oberste Schicht realisiert die funktionalen Anforderungen eines Workflow-Management-Systems. Dort werden die nichtfunktionalen Anforderungen möglichst wenig berücksichtigt und auf die unteren Schichten, auch Middleware-Schichten genannt, verlagert. Durch die Schichtenarchitektur wird die Komplexität eines Workflow-Mangagement-Systems in mehrere we-

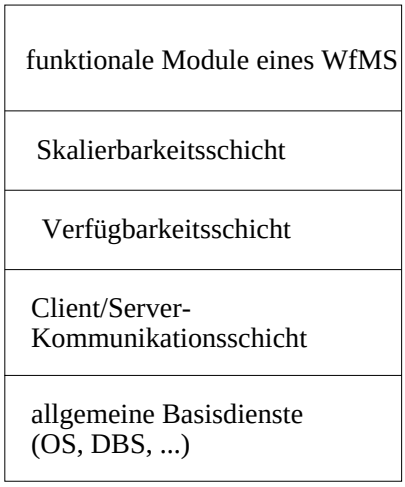

Abbildung 3.7: Schichtenarchitektur eines Workflow-Management-Systems [JBS97, S. 241]

niger komplexe Teilprobleme zerlegt, und Lösungen für einzelne Schichten sind austauschbarer geworden.

Weitere wichtige Forderungen sind die parallele Verarbeitung von Workflows zur Performanzsteigerung und Fehlertoleranz. Daraus ergibt sich der Bedarf nach Koordination wie z. B. Transaktionskonzepte. Das bei Datenbanken verwendete ACID-Transaktionsprinzip (Atomarität, Konsistenz, Isolation, Dauerhaftigkeit, siehe [HS95, S. 390]) muß für die transaktionsbasierte Abarbeitung von Workflows aufgeweicht werden. Geschachtelte Transaktionen entsprechen eher der hierarchischen Struktur von Workflows. Das Alles-oder-Nichts-Prinzip ist oft nicht erwünscht, es darf nicht der gesamte Workflow zurückgesetzt werden, weil eine Teiltransaktion scheitert. Die Kompensation zur Aufhebung abgeschlossener Transaktionen ist notwendig. Isoliertheit wird teilweise aufgehoben, da eventuell mehrere Workflows sich gemeinsame Ressourcen teilen sollen, z. B. bei langen Ausführungszeiten von Workflows oder kooperativer Gruppenarbeit auf denselben Daten. Weiterhin sind Recoverymechanismen notwendig, um nach Systemfehlern einen konsistenten Workflow-Zustand zu rekonstruieren.

Kapitel 4

Das Mini-Workflow-Management-System

In diesem Kapitel wird das Workflow-Management-System vorgestellt. Als Basis diente das in der Studienarbeit [Goh99] implementierte Workflow-Management-System, mit dem eine Studien- und Diplomarbeitsverwaltung mit Anbindung an das WWW realisiert wurde.
Der Abschnitt 4.1 beschreibt den konzeptuellen Aufbau des zugrundeliegenden Workflow-Metaschemas. In Abschnitt 4.2 erfolgt die Vorstellung der konkreten Implementierungsarchitektur unter Verwendung des Ereigniskonzeptes des Datenbank-Management-Systems *OpenIngres*. Anschließend wird in Abschnitt 4.3 die Funktionsweise der Workflow-Instanzausführung erläutert. In Abschnitt 4.4 wird die Kommunikation mit den externen Applikationen beschrieben. Der letzte Abschnitt des Kapitels gibt einige abschließende Bemerkungen wieder. In Anhang C ist die Dokumentation zur Implementation des Mini-WfMS zu finden.

4.1 Das Workflow-Metaschema

Nach [JBS97, S. 123] ist ein Workflow-Metaschema folgendermaßen aufgebaut. Im Workflow-Metaschema sind für jeden zu realisierenden Aspekt Konstrukte festgelegt, die die Workflow-Sprache bilden. Weiterhin müssen die Beziehungen zwischen den Konstrukten dargestellt und Konsistenzbedingungen formuliert sein, um Workflow-Schemata mit einem korrekten strukturellen Aufbau definieren zu können. Benötigte Beziehungstypen sind die Teil-/Ganzes-Beziehung (part-of), die Schema-/Instanzbeziehung (is-instance-of), die Schema-/Subschema-Beziehung (is-a) und Verwendungsbeziehung (plays-role). In den Konsistenzbeziehungen werden Einschränkungen formuliert, denen Beziehungen und Bezeichner

entsprechen müssen. Beispiele sind die Festlegung von Primär- und Fremdschlüsseln.

Die Ausführung eines so konstruierten Workflow-Schemas muß durch eine Ablaufsemantik festgelegt werden. Diese Ablaufsemantik kann implizit oder explizit bereitgestellt sein. *Implizit* bedeutet, daß die Semantik fest im Workflow-Management-System codiert ist. *Explizit* bedeutet hingegen, daß die Ablaufsemantik entweder im Workflow-Metaschema spezifiziert ist und für alle Workflow-Schemata gilt, oder jedes Workflow-Schema sogar seine eigene Semantik festlegt, die nur für Workflow-Instanzen dieses Schema gelten.

4.1.1 Die Struktur eines Workflow-Schemas

Der strukturelle Aufbau des Workflow-Metaschemas gliedert sich in zwei Teile, der Definition der Workflow-Sprache und die Beschreibung der Beziehungen zwischen den Sprachkonstrukten. Die Grammatik zur Workflow-Sprache mit einer Erläuterung der Konstrukte ist in Abschnitt 4.1.2 zu finden.

Bei der Entwicklung der Workflow-Sprache habe ich mich stark an der in [JBS97, S. 176] vorgestellten Workflow-Sprache MSL (Mobile Script Language) des Workflow-Management-Systems MOBILE orientiert. Dort erfolgt eine klare Trennung zwischen den Sprachkonstrukten der einzelnen Aspekte (siehe Abschnitt 3.3.3) im Workflow-Schema. Die Workflow-Sprache MSL ist blockorientiert, jeder Aspekt ist in seinem eigenen Block innerhalb des Workflow-Schemas definiert. Dadurch bleibt das Workflow-Schema modular und erweiterbar. Dies findet sich auch in meiner Workflow-Sprache wieder. Eine Einschränkung ist der Datenfluß, der durch eine Parametrisierung der Kontrollflußkonstrukte dargestellt wird. Prinzipiell ist dann ein Workflow-Schema wie folgt aufgebaut:

```
WORKFLOWTYPE <Workflow-Schemaname>
  SUBWORKFLOWS
    # Funktionsaspekt: Subworkflow-Namen
  END_SUBWORKFLOWS
  WORKFLOW_CONSTANTS
    # Informationsaspekt: workflow-lokale Konstanten
  END_WORKFLOW_CONSTANTS
  WORKFLOW_VARIABLES
    # Informationsaspekt: workflow-lokale Variablen
  END_WORKFLOW_VARIABLES
  OPERATIONS
    # Funktionsaspekt: Operationen
    # Informationsaspekt: Datenfluß
```

```
    # Operationsaspekt: Anbindung externer Programme
  END_OPERATIONS
  CONTROL_FLOW
    # Verhaltensaspekt: Kontrollfluß
    # Informationsaspekt: Datenfluß
  END_CONTROL_FLOW
  ORGANISATIONAL_POLICY
    # Organisationsaspekt
  END_ORGANIZATIONAL_POLICY
END_WORKFLOW_TYPE
```

Die Definition des Workflow-Schemas ist unabhängig von ihrer Verwendung. So kann ein Workflow-Schema in einem beliebigen anderen zur Definition eines Subworkflows genutzt werden. Desweiteren sind Schemata für Applikationstypen definierbar, die die externen Programme anbinden. Alle Schemata werden in Tabellen einer relationalen Datenbank, hier OpenIngres, gespeichert. Bei der Workflow-Instanziierung erfolgt dann die Interpretation der Sprachkonstrukte des gespeicherten Workflow-Schemas und deren Ausführung. Kontrollflußschemata mit Semantik sind im Gegensatz zu MSL nicht zusätzlich definierbar. Der Einfachheit halber wurde eine Anzahl benötigter Konstrukte fest vorgegeben, welche Sequenz, Parallelität, bedingte Verzweigungen und Schleifen abdecken. Deren Semantik ist im Interpretermodul fest implementiert. Dasselbe gilt für Schemadefinitionen von Datentypen. Zur Vereinfachung gibt es nur wenige einfache Datentypen zur Typisierung von Workflow-Variablen und Workflow-Parametern.

In Abbildung 4.1 auf der nächsten Seite ist das dem Workflow-Management-System zugrundeliegende ER-Modell abgebildet, welches die Beziehungen zwischen den einzelnen Aspekten und deren Konstrukten darstellt.

4.1.2 Die Grammatik der Workflow-Sprache und die Bedeutung der Konstrukte

Zur Definition der Grammatik der Workflow-Sprache wurde die *Erweiterte Backus-Naur-Form*[1] verwendet.

Das Workflow-Metaschema umfaßt alle möglichen Konstrukte der Workflow-Sprache.

[1] Der Syntax der EBNF ist nachlesbar in [AL91, S. 54]

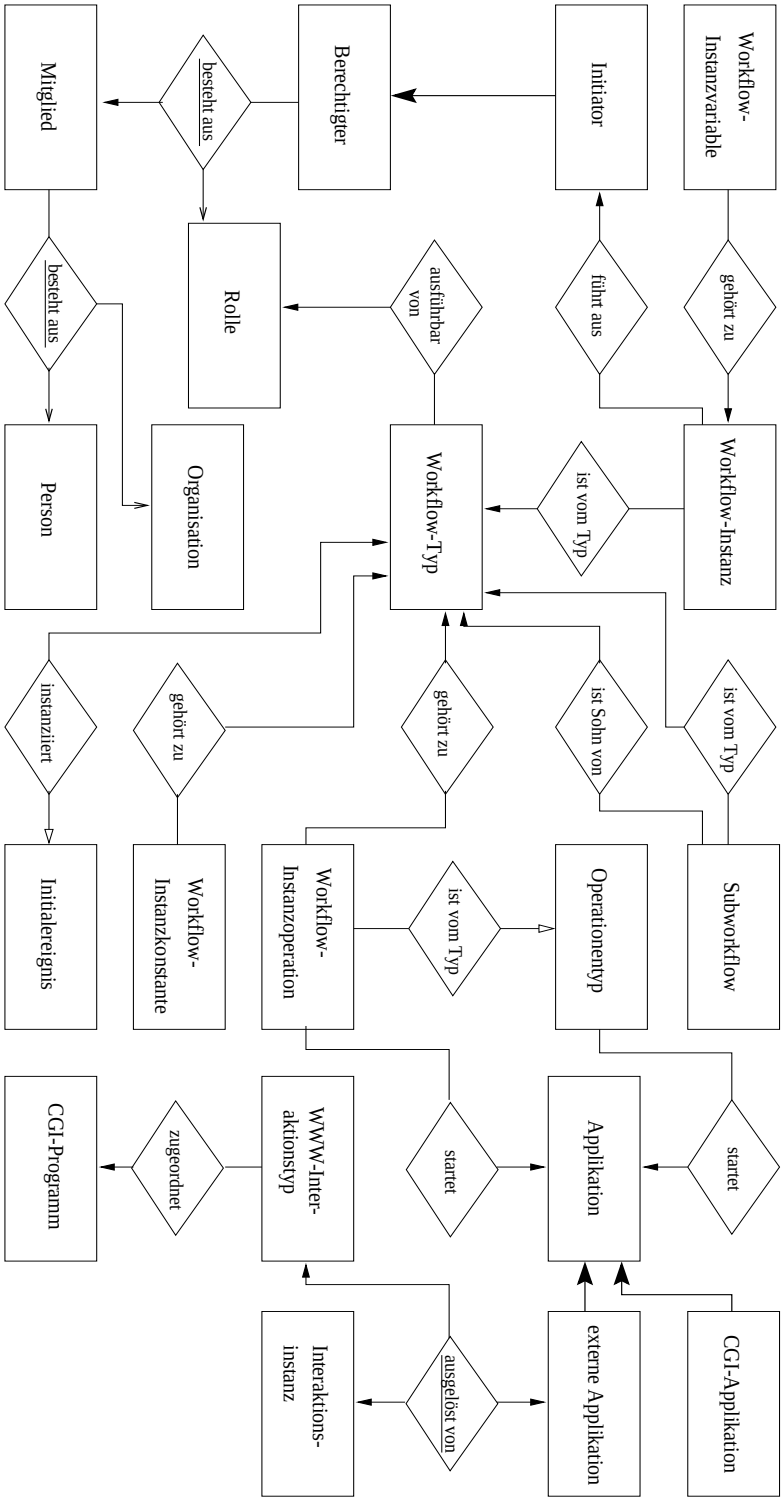

Abbildung 4.1: ER-Modell des Mini-WfMS

```
Workflow-Metaschema =   Workflow-Schema
                        Operationstyp
                        Rolle
                        Ereignis .
```

Im Workflow-Schema sind die Blöcke der einzelnen Aspekte angeordnet.

```
Workflow-Schema =     "WORKFLOWTYPE" Workflowtypname
                      [ Subworkflow-Block ]
                      [ Workflow-Konstanten-Block ]
                      [ Workflow-Variablen-Block ]
                      [ Operations-Block ]
                      Kontrollflußblock
                      Organisations-Block
                      ["END_WORKFLOWTYPE"] .
```

```
Workflowtypname = Bezeichner .
Bezeichner = Buchstabe { ( Buchstabe | Ziffer | "_" ) } .
```

Bei Bezeichnern wird nicht zwischen Groß- und Kleinschreibung unterschieden. Die Möglichkeit dazu dient nur der besseren Lesbarkeit des Workflow-Schemas.

```
Konstante = ( Zahl | Zeichenkette ) .
Zeichenkette = { ( Buchstabe | Ziffer | Sonderzeichen ) } .
Zahl = [ "-" ] Ziffer { Ziffer } .
Buchstabe = ( kleiner_Buchstabe | grosser_Buchstabe ) .
kleiner_Buchstabe = ( "a" | "b" | ... | "z" ) .
grosser_Buchstabe = ( "A" | "B" | ... | "Z" ) .
Ziffer = ( "0" | ... | "9" ) .
Sonderzeichen= ( "!" | " " | ... ) .
```

4.1.2.1 Konstrukte des Kontrollflußaspektes

Im Kontrollflußblock wird der Verhaltensaspekt des Workflows implementiert. Er ist unbedingt erforderlich.

```
Kontrollflußblock =   "CONTROL_FLOW"
                         Kontrollflußkonstrukt
                         { Kontrollflußkonstrukt }
                      "END_CONTROL_FLOW" .
```

Der Kontrollflußblock setzt sich aus Kontrollflußkonstrukten zusammen, die fest vorgegeben sind.

```
Kontrollflußkonstrukt = ( Workflow-Startanweisung |
                          Operation-Startanweisung |
                          If-Anweisung |
                          While-Anweisung |
                          Until-Anweisung |
                          Wertzuweisung |
                          Return-Anweisung |
                          Kommentar-Anweisung
                        ) ";" .
```

Die Anweisungen im einzelnen:

```
Workflow-Startanweisung = "START_WF [" Subworkflow
   { "," Subworkflow "]" .

Subworkflow = Subworkflow-Variable [ "(" Parameter
   [ "," Listenelement-Variable ] ")"
   [ "FOREACH" Listenelement-Variable
     "FROM" Workflow-Listenvariable
     "SUCCESS BY" ( "ANY" | "ALL" ) ] ] .

Subworkflow-Variable = Bezeichner .

Workflow-Listenvariable = Workflow-Variable .

Listenelement-Variable = Workflow-Variable .

Parameter = [ ( Workflow-Variable | Workflow-Konstante )
   { "," ( Workflow-Variable | Workflow-Konstante ) } ] .

Workflow-Variable = Bezeichner .

Workflow-Konstante = Bezeichner .
```

Die Workflow-Startanweisung dient zum Starten von Subworkflows mit Workflow-Variablen oder -Konstanten als Parameter. In einer Anweisung gestartete Sub-

workflows werden parallel ausgeführt. Die weitere Abarbeitung des Kontrollfluß-blockes wird solange ausgesetzt, bis alle gestarteten Subworkflows beendet wurden. Eine Sequenz kann also mit nacheinander stehenden Workflow-Startanweisungen implementiert werden. Das FOREACH-Konstrukt der Startanweisung startet für jedes Listenelement der Listenvariable den zugehörigen Subworkflow, an den die angegebenen Parameter und das jeweilige Listenelement übergeben werden. Dabei wird der Erfolgsstatus des Subworkflows nach Ausführung der Subworkflow-Instanzen entweder durch eine Oder-Verknüpfung ("ANY") oder durch eine UND-Verknüpfung ("ALL") aller Statuswerte der Subworkflow-Instanzen gebildet. Die Parameterübergabe erfolgt als Wertübergabe über die Bezeichnergleichheit, d. h. der Wert eines Parameters wird einer Subworkflow-Variablen gleichen Namens zugewiesen. Diese muß im Subworkflow-Schema definiert sein, die Typgleichheit zwischen Workflow- und Subworkflow-Variable sichert der Schemaentwickler. Generell gilt: Alle benutzten Variablen und Konstanten in einem Workflow-Schema müssen definiert sein.

```
Operations-Startanweisung ="START_OP" Operationsname .

Operationsname = Bezeichner .
```

Die Operations-Startanweisung startet eine Operation, die eine externe Applikation ausführt. Die Operation muß im Operationsblock des Workflow-Schemas definiert sein. Die Abarbeitung des Kontrollflusses wird ausgesetzt, bis die Operation beendet wurde. Nach der Ausführung wird in einer Steuervariable des Workflows, die den gleichen Namen wie die Operation hat, der Abarbeitungsstatus gespeichert.

```
If-Anweisung = "IF" "(" Ausdruck ")" "THEN"
   "{" { Kontrollflußkonstrukt } "}"
   [ "ELSE" "{" { Kontrollflußkonstrukt } "}" ]
   "ENDIF" .

Ausdruck = ( Element |
   ( Element | "(" Ausdruck ")" ) binärer_Operator
   ( Element | "(" Ausdruck ")" ) |
   unärer_Operator "(" Element | Ausdruck ")" ) .

binärer_Operator = ( "AND" | "&&" | "OR" | "||" ) .

unärer_Operator = "NOT" .

Element = ( Workflow-Variable | Zahl | "'"Zeichenkette"'" ) .
```

Die If-Anweisung arbeitet in bekannter Weise. Ist der Ausdruck wahr, wird der THEN-Zweig ausgeführt, ansonsten der optionale ELSE-Zweig. Der Wahrheitswert eines Elementes ist 0 (*false*) für die leere Zeichenkette, die Zeichenkette "0"

und die Zahl 0, sonst 1 (*true*). Besteht ein Element aus einer Workflow-Variable, so wird diese vor der Bedingungsauswertung durch den ihr zugewiesenen Wert ersetzt.

```
While-Anweisung = "WHILE" "(" Ausdruck ")"
   "{" { Kontrollflußkonstrukt } "}" "ENDWHILE" .
```

Die While-Anweisung funktioniert ebenfalls in üblicher Weise. Ist der Ausdruck wahr, wird der Schleifenkörper ausgeführt und nach Beendigung wieder zurück zur Bedingungsprüfung gesprungen. Ist er falsch, wird die Abarbeitung des Kontrollflusses nach der While-Anweisung fortgesetzt.

```
Until-Anweisung = "WHILE ONCE" "(" Ausdruck ")"
   "{" { Kontrollflußkonstrukt } "}" "ENDWHILE" .
```

Die Until-Anweisung führt den Schleifenkörper einmal aus, danach wird der Ausdruck geprüft. Ab jetzt funktioniert sie wie die While-Anweisung.

```
Wertzuweisung = "SET" Workflow-Variable
   ( "(" Konstante ")" | CALCULATE "(" Ausdruck ")" ) .
```

Mit dieser Anweisung kann einer Workflow-Variablen ein Wert zugwiesen werden. Das ist entweder ein konstanter oder ein durch das `CALCULATE`-Konstrukt aus dem Ausdruck berechneter Wert.

```
Return-Anweisung = "RETURN" [ "(" Parameter ")" ]
   [ "WITH STATUS" "(" Ausdruck ")" ] .
```

Die Return-Anweisung beendet die Ausführung eines Workflows. Die Parameterrückgabe erfolgt wieder als Wertübergabe an eine Superworkflow-Variable gleichen Namens. Diese Variable muß im Superworkflow definiert sein, um Typgleichheit kümmert sich der Entwickler. Desweiteren gibt die Anweisung einen Status über den Erfolg des Workflows an eine Steuervariable im Superworkflow zurück. Diese Variable hat den Namen des Subworkflows. Der Status wird durch eine Konjunktion über alle Steuervariablen berechnet, die den Subworkflows und Operationen des Workflows entsprechen. Diese boolschen Variablen werden zu Beginn der Workflow-Ausführung mit 1 initialisiert, bei Scheitern auf 0 und bei Erfolg auf 1 gesetzt. Damit bleibt ein nicht ausgeführter Subworkflow oder eine nicht ausgeführte Operation bei der Statusberechnung standardmäßig unberücksichtigt. Wird eine andere Berechnung des Erfolgstatus gewünscht, so kann über den Ausdruck in der optionalen `WITH STATUS`-Klausel eine andere Berechnungsvorschrift angegeben werden.

```
Kommentar-Anweisung = "#" Zeichenkette .
```

Eine Kommentaranweisung muß in einer eigenen Zeile stehen. Kommentare werden beim Einbringen des Workflow-Schemas in die Datenbank entfernt.

Alle Bezeichner in einem Workflow-Schema sollten eindeutig sein, egal wofür sie stehen. Bezeichner sollten nicht mit einem Unterstrich anfangen, da mit diesem Zeichen interne Steuervariablen des Workflows beginnen.

4.1.2.2 Konstrukte des Funktionsaspektes

Definition der Subworkflows

Im Subworkflow-Block werden die Subworkflow-Variablen definiert und somit die Funktionsstruktur des Workflows.

```
Subworkflow-Block = "SUBWORKFLOWS"
                    {Subworkflow-Variable":"Workflowtypname}
                    "END_SUBWORKFLOWS" .
```

Definition der Workflow-Operationen

Im Operationsblock des Workflow-Schemas können eigene Operationen zur Einbindung externer Programme definiert werden. Diese sind lokal gültig und können nur von Workflows dieses Workflow-Schemas genutzt werden.

```
Operations-Block =    "OPERATIONS"
                      { Operation }
                      "END_OPERATIONS" .
```

Dabei gibt es zwei Operationsarten, abgeleitete und selbständige Operationen.

```
Operation = ( abgeleitete_Operation |
              selbständige_Operation ) .
```

Die selbständige Operation realisiert die Anbindung eines externen Programmes für ein Workflow-Schema, ohne Wiederverwendbarkeit zu bieten.

```
selbständige_Operation ="OPERATION" Operationsname
                            [ "IN" Parameter
                            [ "OUT" Workflow-Variable { "," 
                                    Workflow-Variable } ]
                            "OPERATIONBODY"
                               Operationsimplementierung
                            "END_OPERATIONBODY"
                          "END_OPERATION" .


Operationsimplementierung = ("RUN_FILE:" Programm) |
                              ("HTTP_REDIRECT:" Programm) .

Programm = gültiger_Dateiname .
```

In der Operationsimplementierung wird die Art und Weise des Aufrufes einer externen Applikation durch das Workflow-Management-System festgelegt. Diese Applikation ist dann in der Regel eine Workflow-Applikation, in der das externe Programm eingebettet ist. Sie kümmert sich um die Kommunikation mit dem Workflow-Management-System. Die konkrete Vorgehensweise beim Start der Applikation und der Parameterüber- und rückgabe wird durch das `RUN_FILE`- bzw. das `HTTP_REDIRECT`-Konstrukt bestimmt. Mit dem `HTTP_REDIRECT`-Konstrukt erfolgt die Anbindung von CGI-Programmen an die WWW-Schnittstelle. Das `RUN_FILE`-Konstrukt dient zur Ausführung von Programmen, die keine Interaktion mit dem Nutzer über die WWW-Schnittstelle erfordern, oder aber erst bei Bedarf eine Nutzerinteraktion anfordern. Dabei ist zu bemerken, daß die CGI-Programme auf dem WWW-Server laufen, während die Programme, die über das `RUN_FILE`-Konstrukt gestartet wurden, auf dem Rechner des Workflow-Management-Systems ausführt werden. Als Übergabeparameter erhält die Workflow-Applikation die Variablenbezeichner aus dem `IN`-Konstrukt der Operation mit den zugehörigen Werten. Als Rückgabeparameter erhält das Workflow-Management-System Variablenbezeichner-Wert-Paare von der Workflow-Applikation. Die Werte werden dann den nach dem `OUT`-Konstrukt stehenden Workflow-Variablen gleichen Namens zugewiesen. Details zur programmtechnischen Realisierung sind in Abschnitt 4.4 nachlesbar.

Die abgeleitete Operation greift auf einen global definierten Operationstyp zurück, in dem die Workflow-Applikation gekapselt ist. Dieser steht für alle Workflow-Schemata zur Verfügung.

```
abgeleitete_Operation = "OPERATION" Operationsname
                        "OF" "OPERATIONTYPE" Operationstyp-
                        name
                         [ "IN" Parameter
                         [ "OUT" Workflow-Variable { "," 
                                 Workflow-Variable } ]
                         "OPERATIONBODY"
                           { Operationsvariable "="
                           ( Zahl | Zeichenkette ) }
                         "END_OPERATIONBODY"
                        "END_OPERATION" .

Operationstypname = Bezeichner .

Operationsvariable = Bezeichner .
```

Über das IN-Konstrukt wird festgelegt, welche Workflow-Variablen an den Operationstyp übergeben werden. Im OPERATIONBODY können ebenfalls Eingabevariablen definiert werden, deren Werte unabhängig von Workflow-Variablen sind. Durch diese Kapselung bleibt das restliche Workflow-Schema unberührt, es ist Sache der lokalen Operation, weitere Einstellungen für die externen Applikation vorzunehmen. Hinter dem OUT-Konstrukt stehen Workflow-Variablen, die die Rückgabewerte des Operationstyps aufnehmen. Die Namen der Ein- und Ausgabevariablen müssen mit den im Operationstyp vorgegebenen Bezeichnern übereinstimmen, da die Wertzuweisung über gleiche Bezeichnernamen erfolgt.

Über Operationstypen können allen Workflow-Schemata externe Applikationen zur Verfügung gestellt werden.

```
Operationstyp = "OPERATIONTYPE" Operationstypname
                  [ "IN" Variablenname { "," 
                         Variablenname } ]
                  [ "OUT" Variablenname { "," 
                          Variablenname } ]
                  "OPERATIONBODY"
                     Operationsimplementierung
                  "END_OPERATIONBODY"
                  [ Beschreibung ]
                "END_OPERATIONTYPE" .

Variablenname = Bezeichner .
```

```
Beschreibung = "DESCRIPTION"
                  { Zeichenkette }
               "END_DESCRIPTION" .
```

Nach dem IN-Konstrukt stehen festgelegte Variablennamen, die die abgeleiteten Operationen zur Übergabe der Eingabeparameter benutzen müssen. Die abgeleitete Operation kann Eingabeparameter, falls nicht benötigt, weglassen. Welche Variablen tatsächlich übergeben werden, steht bei der Definition der abgeleiteten Operation im Workflow-Schema. Hinter dem OUT-Konstrukt stehen Bezeichner für Workflow-Variablen, in die das Workflow-Management-System die über den Bezeichnernamen korrespondierenden Rückgabewerte der externen Applikation ablegt. Auch hier können Rückgabeparameter bei der Definition der abgeleiteten Operation weggelassen werden, je nachdem, was benötigt wird. Im DESCRIPTION-Teil kann die Bedeutung der Variablenbezeichner und des externen Programms textuell erklärt werden.

4.1.2.3 Konstrukte des Informationsaspektes

In den nachstehenden Blöcken werden Workflow-Konstanten und Workflow-Variablen definiert. Sie werden für den Datenfluß der Nutzdaten innerhalb des Workflows benutzt. Interne Steuervariablen des Workflows werden hier nicht definiert, jeder Workflow bekommt diese automatisch zugewiesen.

Einer Workflow-Konstante wird im Workflow-Schema ein konkreter Wert zugewiesen, der sich nicht ändern läßt. Damit kann man Workflow-Instanzen mit einer Datenbasis auszustatten, zum Beispiel mit einen Umrechnungskurs.

```
Workflow-Konstanten-Block ="WORKFLOW_CONSTANTS"
                           { Workflow-Konstante ":"
                           Variablentyp "("Konstante")" }
                           "END_WORKFLOW_CONSTANTS" .

Workflow-Konstante = Bezeichner .
Variablentyp = ( Standardtyp | "list(" Variablentyp ")" ) .
Standardtyp = ( "STRING" | "BOOLEAN" | "NUM" ) .

Workflow-Variablen-Block = "WORKFLOW_VARIABLES"
                           {["IN:"] Workflow-Variable ":"
                           Variablentyp ["("Konstante")"]}
                           "END_WORKFLOW_VARIABLES" .
```

Das IN-Konstrukt bei den Workflow-Variablen hat nur eine syntaktische Funktion. Damit lassen sich zur besseren Übersichtlichkeit Variablen auszeichnen, die durch Eingabeparameter belegt werden.

4.1.2.4 Konstrukte des Organisationsaspektes

Im Organisationsblock steht eine Berechtigung, wer den Workflow ausführen darf, und das Initialereignis, welches zum Start führen soll. Ein Initialereignis wird einem Superworkflow zugeordnet, der einem bestimmten Arbeitsvorgang entspricht. Weiterhin kann eine aussagekräftige Bezeichnung sowie eine Beschreibung zum Workflow-Typ angegeben werden.

```
Organisations-Block =   "ORGANISATIONAL_POLICY"
                           Berechtigung
                           [ Initial-Ereignis ]
                           [ Workflowtyp-Aliasname ]
                           [ Beschreibung ]
                        "END_ORGANIZATIONAL_POLICY" .

Berechtigung = "ROLE:" Rollenname .

Rollenname = Bezeichner .

Initial-Ereignis = "INITIAL_EVENT:" Ereignisname .

Ereignisname = Bezeichner .

Workflowtyp-Aliasname = "PUBLIC_NAME:" Zeichenkette .
```

Einer Person können Rollen zugewiesen werden. Dadurch erlangt sie die Berechtigung, Workflows zu starten, die mit der Rolle gekennzeichnet sind. Diese Zuordnungstabellen sind in der Datenbank gespeichert und werden zur Laufzeit abgerufen.

```
Rolle =   "ROLENAME:" Rollenname
          [ Beschreibung ] .
```

Ist einem Ereignis ein Workflow-Schema zugeordnet, nennt man es Initial-Ereignis. Diese Zuordnungen sind in der Datenbank gespeichert und werden bei Auftritt eines Ereignisses abgefragt. Es erfolgt die Instanziierung des Workflow-Schemas und die Ausführung der Workflow-Instanz.

```
Ereignis =   "EVENTNAME:" Ereignisname
             [ Beschreibung ] .
```

```
Ereignisname = Bezeichner .
```

Beispiele für Workflow-Schemata sind in Anhang A.1 zu finden.

4.2 Implementierungsarchitektur mit Ereignismanager

Das zentrale Element in der Workflow-Steuerung ist der *Ereignismanager*. Über diesen wird die Workflow-Ausführung gesteuert. Der Nutzer kommuniziert mit der *Anwendungsapplikation* über die *WWW-Schnittstelle*. Sie wird durch ein CGI-Programm realisiert, dessen generierte HTML-Seiten von einem HTML-Browser angezeigt werden. Die Anwendungsapplikation löst ein Initialereignis aus, um einen Workflow zu starten. Der Ereignismanager reagiert auf das auftretende Ereignis mit der entsprechenden Behandlungsroutine. So wird die Instanziierung eines Workflow-Schemas angestoßen und die Ausführung der Workflow-Instanz gestartet. Superworkflows werden über das Ende von Subworkflows benachrichtigt und beendete Workflow-Instanzen aus der Instanztabelle gelöscht. Das *Instanziierungsmodul* erstellt Workflow-Instanzen in der Workflow-Instanztabelle und initialisiert deren Datenbereiche. Das *Interpretermodul* führt Workflow-Instanzen aus und startet Operationen. Das *Operationsmodul* ist für den Start und die Kommunikation mit *Workflow-Applikationen* zuständig, welche die *externen Programme* oder *CGI-Programme* anbinden. Dabei werden vom Workflow-Management-System Programmbibliotheken bereitgestellt, die die Workflow-Applikationen zur Anbindung benutzen. Die Administrationswerkzeuge dienen zum Einbringen der Workflow-Schemadefinitionen in die Datenbank und zur Überwachung. Unter Einbeziehung der Ereignisbehandlung ergibt sich die in Abbildung 4.2 auf der nächsten Seite abgebildete Implementierungsarchitektur des Workflow-Management-Systems.

4.3 Die Ausführung einer Workflow-Instanz

Ein Workflow-Schema wird bei Eintritt des zugehörigen Initialereignisses durch das Initialisierungsmodul des Workflow-Management-Systems instanziiert. Die Ereignisbehandlung erfolgt durch den Ereignismanager. Bei der Instanziierung wird die Definition des Workflow-Schemas aus der Datenbank geladen, und es erfolgt die Initialisierung der definierten Workflow- und Steuervariablen. Vorhandene Initialisierungsparameter des Initialereignisses werden übernommen, so zum Beispiel der Auslöser des Initialereignisses, der Initiator. Die Instanz wird in die

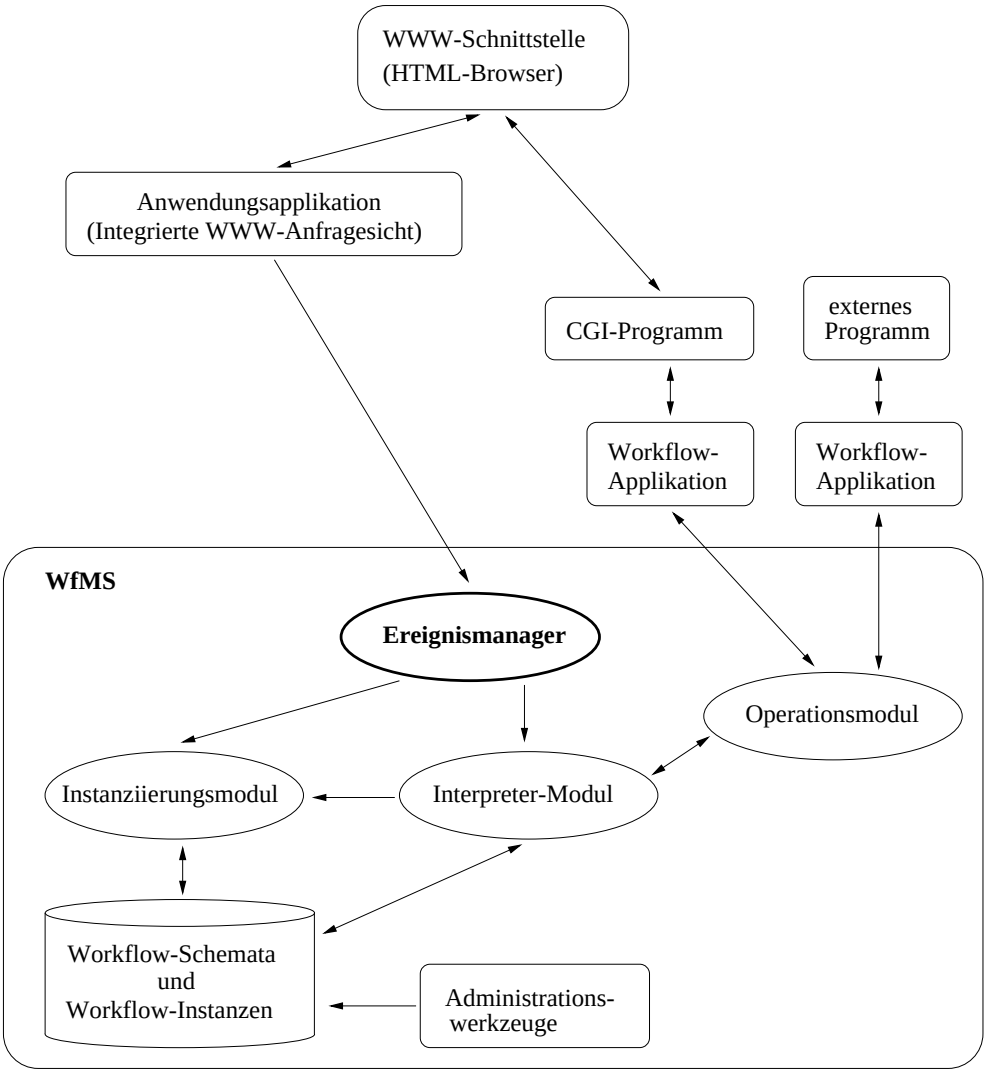

Abbildung 4.2: Implementierungsarchitektur des Mini-WfMS

Workflow-Instanztabelle mit einem eindeutigen Schlüsselwert eingetragen. Sie bekommt einen Status zugewiesen, der den Zustand der Workflow-Instanz kennzeichnet, hier den Status NEW. Ein Trigger, der das Statusfeld überwacht, löst ein Ereignis aus, das als Parameter den neuen Status und den Workflow-Instanz-Schlüssel enthält. Der Ereignismanager erkennt das Ereignis und führt die entsprechende Ereignisbehandlungsroutine aus. Dadurch erfolgt der Start des Interpretermoduls, welches mit der Abarbeitung des Kontrollflusses der Workflow-Instanz beginnt und so die Kontrolle über die Ausführung übernimmt. Der Status der Workflow-Instanz wird auf RUNNING geändert. Kontrollflußanweisung nach Kontrollflußanweisung wird interpretiert und die Implementation der dazu gehörigen Ablaufsemantik ausgeführt. Das Beenden der Abarbeitung bewirkt die `RETURN`-Anweisung. Falls die Workflow-Instanz ein Subworkflow ist, werden mögliche Rückgabeparameter an den Superworkflow übergeben. Der Workflow-Instanz-Status erhält den Wert END und den Erfolgsstatus des Workflows. Der Ereignismanager empfängt das durch einen Trigger ausgelöste Ereignis und erledigt notwendige Nacharbeiten wie das Löschen der Workflow-Instanz aus der Datenbank, Informieren des Superworkflows bzw. des Initiators über das Workflow-Ende.

Soll ein Subworkflow gestartet werden, übernimmt das Initialisierungsmodul die Instanziierung des zugehörigen Workflow-Schemas, dessen Definition wird geladen, die Workflow- und Steuervariablen initialisiert und Übergabeparameter übernommen. Dann erfolgt der Eintrag in die Workflow-Instanztabelle. Jetzt muß der Ausführende des Subworkflows bestimmt werden. Das geschieht über die dem Workflow-Schema zugewiesene Rolle, die ausgewählte Bearbeiter spielen dürfen, und der Selektion eines Bearbeiters. Im Moment erfolgt die Bestimmung des Ausführenden, also dem späteren Initiator des Subworkflows, sehr rudimentär. Hat der Initiator des Superworkflows auch die Berechtigung, den Subworkflow auszuführen, wird er Initiator des Subworkflows. Die Belegung des Statusfeldes in der Workflow-Instanztabelle erfolgt mit NEW, dadurch beginnt die sofortige Ausführung des Subworkflows. Ansonsten wird der erste gefundene, zur Ausführung des Subworkflows berechtigte Bearbeiter Initiator des Subworkflows. Der Subworkflow erhält den Status WORKLIST, d.h. er wird in die Arbeitsliste des Bearbeiters eingetragen, der den Workflow dann später ausführt.

Um eine bessere Steuerung der Bearbeiterauswahl zu unterstützen, muß der Organisationsaspekt erweitert werden. Die Bestimmung eines Bearbeiters könnte durch eine Auswahlfunktion, die zu einem Subworkflow-Aufruf angegeben wird, erfolgen. Damit kann die Auswahl den Erfordernissen angepaßt werden. Die Auswahlfunktionen können global (benutzbar für alle Workflow-Schemata) sowie lokal (nur in einem Workflow-Schema benutzbar) definiert sein.

Der Superworkflow wird durch den Ereignismanager vom Ende des Subwork-

flows informiert und setzt dann seine Abarbeitung fort. Durch Angabe eines Zeitlimits kann die Abarbeitung vor Ende des Subworkflows fortgesetzt werden, wobei der Subworkflow als fehlgeschlagen gewertet wird.

Zum Start einer Operation wird ein Hilfsmodul ausgeführt, das die Operationsdefinition lädt und Übergabeparameter für das externe Programm zusammenstellt. Dann wird die entsprechende Workflow-Applikation gestartet, welche die Anbindung des externen Programms übernimmt. Die Workflow-Applikation gibt die Parameter an das externe Programm weiter und leitet Rückgabeparameter ans Operationsmodul zurück. Das weist die Rückgabeparameter den entsprechenden Workflow- und Steuervariablen zu.

4.4 Die Kommunikation mit den externen Applikationen

Im Moment können zwei Arten von externen Applikationen angebunden werden. Das sind zum einen CGI-Programme, die mit dem WWW-Browser des Anwenders kommunizieren, und andere externe Programme ohne direkte Interaktionsmöglichkeit mit dem Nutzer.

Die Kommunikation zwischen dem Workflow-Management-System und den CGI-Programmen erfolgt über *Internet-Domainsockets*[2], das ist eine Kommunikationsart zwischen Prozessen. Dadurch können die CGI-Programme und das Workflow-Management-System auf unterschiedlichen Maschinen laufen, was den HTTP-Server entlastet. Nach dem Start eines Workflows durch den Anwender über die WWW-Schnittstelle geschieht folgendes: Die Anwendungsapplikation, ein CGI-Programm, löst ein Initialereignis mit Parametern aus. Die Parameter beinhalten die Nutzerinformationen und die Adresse, auf der das CGI-Programm auf Antwort vom Workflow-Management-System wartet. Der zugehörige Workflow wird ausgeführt, und schließlich startet eine Operation. Dabei verbindet sich das Workflow-Management-System mit dem wartenden Programm unter der angegebenen Adresse. Mit den vom Workflow übermittelten Daten der auszuführenden Operation erfolgt eine Umleitung (HTTP-Redirect) auf das entsprechende CGI-Programm, das der Operation zugeordnet ist. Dieses übernimmt die weitere Interaktion mit dem Anwender. Bei Beendigung werden die Rückgabedaten an die vorher vom Workflow mitgeteilte Adresse gesendet, wo der Workflow auf das Ende der Operation wartet. Dabei wird wieder eine Adresse übermittelt, an die der Workflow die Daten der nächsten Operation senden kann. Das

[2] In [WCS97] wird die Funktionsweise erläutert.

Ende der Workflow-Ausführung sowie deren Erfolgsstatus teilt das Workflow-Management-System der wartenden Anwendungsapplikation mit.

Die andere Möglichkeit zur Anbindung externer Programme wird durch den *Systemaufruf*[3], der einen Kindprozess startet, realisiert. Die Eingabeparameter werden dem externen Programm in der Kommandozeile übergeben und die Rückgabeparameter von der Standardausgabe gelesen. Eine externes Programm kann bei Bedarf eine ihm zugeordnete Interaktion auslösen, die dann mit dem Nutzer über die WWW-Schnittstelle mittels eines CGI-Programmes kommuniziert. Dadurch kann man dem Nutzer die Möglichkeit bieten, zur Lösung eines vom externen Programm festgestellten Konfliktes einzugreifen. Die Operation wird danach mit berichtigten oder ergänzten Eingabedaten neu gestartet.

4.5 Abschließende Bemerkungen

Durch die Modellierung der Arbeitsvorgänge erfolgt eine klare Strukturierung der anfallenden Arbeiten sowie der benötigten Arbeitsmittel und Ressourcen. Besonders gut läßt sich die Ausführungsreihenfolge der Tätigkeiten mit einer Diagrammsprache darstellen. Dies führt zu einer erhöhten Anschaulichkeit des Entwurfes und damit zu einer besseren Verständlichkeit. Die zur Verfügung stehenden Modellierungsmöglichkeiten des Workflow-Management-Systems bestimmen die Einfachheit bzw. sogar die Möglichkeit der Umsetzung der modellierten Arbeitsvorgänge in die späteren Workflows. Deshalb sollte im Idealfall die Auswahl eines Workflow-Management-Systems erst nach abgeschlossener Arbeitsvorgangsmodellierung erfolgen. Da das in der Regel nicht möglich ist, muß man Wert auf die Erweiterbarkeit des Systems hinsichtlich seiner Ausdrucksvielfalt legen. Die Integrierbarkeit des Workflow-Management-Systems in eine bestehende EDV-Infrastruktur und die Anbindung von existierenden Anwendungen sind ebenfalls wichtige Kriterien.

Das hier implementierte Mini-Workflow-Management-System bietet natürlich nur Grundfunktionen. Es wurde nur das implementiert, was auch für die Workflows des Beispiel-Szenarios gebraucht wurde. Das sind im wesentlichen das Analysieren und Ausführen der Kontrollflußkonstrukte des Workflow-Schemas, Parallelität bei der Ausführung der Workflows, die Anbindung von externen Applikationen sowie CGI-Applikationen über die WWW-Schnittstelle als Nutzerschnittsstelle.

Durch die Festlegung auf die WWW-Schnittstelle zur Kommunikation mit dem Anwender ergeben sich sowohl Vor- als auch Nachteile. So wird auf Anwenderseite nur ein Rechner mit HTML-Browser und Intranet/Internet-Anbindung

[3] Zur Erläuterung der Funktionsweise siehe [WCS97].

benötigt, d. h. im Prinzip kann von jedem Rechner mit Internet-Anbindung auf das System zugegriffen werden. Dies ist wichtig, da es als Informationssystem dienen soll und die Benutzung durch den Anwender wesentlich von der Einfachheit des Zugriffes abhängt. Durch die Funktionsweise des HTTP-Protokolls (zustandslos, keine persistente Verbindung zum Server) ist eine vollständige Kontrolle der Workflow-Ausführung auf der Anwenderseite nicht möglich. Der Anwender kann jederzeit in die Nutzerführung mit den Navigationsfunktionen des HTML-Browsers (BACK und FORWARD-Tasten) eingreifen und so zum Workflow inkonsistente Zustände herstellen. Der Einsatz von Javascript, das Abschalten des Browser-Caches oder ein erhöhter Aufwand bei der Anbindung der externen Programme führen nicht zu einer Lösung dieses grundsätzlichen Problems. Um das zu vermeiden, muß man von der WWW-Schnittstelle abrücken und das Workflow-Management-System über eine eigenständige Applikation auf der Client-Seite anbinden. Der Nachteil ist aber eventuell Verlust der Plattformunabhängigkeit, was den Auschluß von Nutzern bedeutet, und ein erhöhter Installations- und damit auch Wartungsaufwand. Trotz dieser Nachteile ist die WWW-Schnittstelle als Anwenderschnittstelle gerade für kleinere Aufgaben geeignet.

Kapitel 5

Integrierte WWW-Anfragesichten - Ein Beispiel

Dieses Kapitel beschäftigt sich mit dem Entwurf und der Implementierung einer integrierten Anfragesicht zu dem Beispiel-Szenario *Reiseverbindungen*. Zuerst werden aber in Abschnitt 5.1 einige grundsätzliche Betrachtungen vorgenommen, um allgemeine Anforderungen an eine integrierte WWW-Anfragesicht darzulegen. Ab Abschnitt 5.2 wird dann die Umsetzung des Beispiel-Szenarios beschrieben.

5.1 Interoperabilität zwischen WWW-Datenquellen

Bei der Integration mehrerer autonomer WWW-Datenquellen treten dieselben Konflikte wie bei föderierten Datenbanksystemen auf. Es existieren schematische und semantische Heterogenitäten, die vom Entwickler erkannt und, wenn erforderlich, behandelt werden müssen. Beispiele für solche Konflikte sind Synonyme und Homonyme bei Attributnamen und -werten sowie unterschiedliche Datengranularitäten, Modellierungsstrukturen und Maßeinheiten. Prinzipiell sind die Föderationstechniken[1] (z. B. feste und lose Kopplung von DBMS) für integrierte Web-Anfragsichten anwendbar, dabei wird die Wahl der Technik durch die Art der benötigten Datenintegration in der Anfragesicht bestimmt.

Zu einer integrierten WWW-Anfragesicht gehören verschiedene Komponenten, die möglichst modular gehalten werden sollten. Im einzelnen sind das:

[1] Eine weitergehende Erläuterung ist in [Con97] zu finden.

- Die *Anfrageschnittstelle* dient zur Formulierung von Anfragen. Die Eingabe erfolgt in der Regel formularbasiert, z. B. über ein HTML-Formular. Das bietet wenig Flexibilität, aber eine integrierte WWW-Anfragesicht ist naturgemäß nur auf wenige Anfragearten spezialisiert.

- Die *Ergebnisrepräsentation* stellt das ermittelte Resultat zu einer Anfrage dar, dies kann je nach der Datenstruktur der Anfragesicht mehr oder weniger aufwendig sein.

- Es werden *Speicherstrukturen* für Zwischen- und Anfrageergebnisse benötigt. Das kann zum Beispiel eine Datenbank oder eine Datei sein.

- Mittels der *Interaktionskomponente* wird dem Nutzer die Möglichkeit zur Auflösung von Konflikten geboten, die vom System erkannt, aber nicht behandelt werden konnten.

- Die voneinander unabhängigen *Wrapper* binden die WWW-Datenquellen an.

- Die *Kontroll- und Datenflußkomponenten* steuern den Ablauf der Anfrage und die Datenflüsse zwischen den Wrappern sowie anderen Komponenten. Diese Aufgabe übernimmt in der Beispiel-Anfragesicht das in Kapitel 4 vorgestellte Workflow-Management-System.

- Weiterhin werden *Hilfskomponenten* benötigt, die automatisch Konflikte zwischen den heterogenen Datenquellen auflösen und Zwischenergebnisse aus gewonnenen Daten berechnen, die zur weiteren Verarbeitung in der Anfrage dienen.

Der modulare Aufbau einer WWW-Anfragesicht erleichtert den Austausch von Komponenten, verbessert Erweiter- und Wartbarkeit und ermöglicht Parallelität bei der Anfrageausführung.

5.2 Das Beispiel-Szenario *Reiseverbindungen*

Die Idee dieses Beispiel-Szenarios ist es, über eine integrierte WWW-Anfragesicht Reiseverbindungen zu ermitteln, die aus den Fahrplänen verschiedener Verkehrsunternehmen zusammengesetzt werden. Um das Szenario in der Komplexität einzuschränken, werden nur zwei Arten von Verkehrsnetzen berücksichtigt. Das ist der landgestützte Verkehr, also Bahn, Bus usw., und der Flugverkehr. Dabei sind die Flughäfen die Verbindungspunkte zwischen den beiden Verkehrsnetzen.

Als WWW-Datenquellen werden die Fahrplanauskunft *Hafas* der Deutschen Bahn (*http://bahn.hafas.de*) und der Fluganbieter *Traveloverland* (*http://www.traveloverland.de*) in die Anfragesicht integriert. Sie ist um andere WWW-Datenquellen (Verkehrsunternehmen) erweiterbar.
Die Fahrplanauskunft Hafas ist eine Verbundsauskunft, d. h. es sind viele Fahrpläne anderer Verkehrsunternehmen integriert und das europaweit. Teilweise sind sogar innerstädtische Straßenbahnverbindungen und Fußwegangaben zwischen Umsteigepunkten erfaßt. In Anhang B ist das Anfrageformular von Hafas (Abbildung B.1) und eine Beispiel-Ergebnisseite mit gefundenen Verbindungen (Abbildung B.2) zu sehen.
Der Anbieter Traveloverland bietet ein Informations- und Buchungssystem für Flugverbindungen an. In Anhang B ist das Anfrageformular des Systems und mehrere Seiten mit Ergebnisdaten zu sehen. Bei Traveloverland erfolgt die Selektion einer Flugverbindung in mehreren Schritten. Zuerst wird zu einer Anfrage eine Ergebnisseite mit nach Preisen geordneten möglichen Flugkategorien ausgegeben. Im weiteren Verlauf wird von Fluggruppen gesprochen. Nach Wahl einer Fluggruppe erhält man eine Liste der zugehörigen Flugverbindungen, zu denen man sich jeweils die Flugdetails sowie Rückflugverbindungen anzeigen lassen kann. Zur Verdeutlichung sind in Anhang B das Anfrageformular (Abbildung B.3) und Beispiele für die einzelnen Ergebniseiten (Abbildungen B.4, B.5, B.6) dargestellt.
Beide WWW-Datenquellen benutzen zur Präsentation nur HTML. Dadurch sind die zu implementierenden HTML-Wrapper nicht im voraus den in Abschnitt 2.4 genannten Einschränkungen bei der Datengewinnung unterworfen. Auch ohne Javascript, Java oder ähnliches wird eine gute Bedienbarkeit, Übersichtlichkeit und Komfort geboten.

5.2.1 Die Anfrageschnittstelle

Die integrierte WWW-Anfragesicht hat ein HTML-Formular (siehe Abbildung 5.1 auf Seite 69) als Anfrageschnittstelle, in dem die nötigen Angaben für die gesuchten Reiseverbindungen eingetragen werden. Das sind im einzelnen:

- der Abfahrtsort mit Abfahrtszeit-Intervall,
- der Abflugshafen mit Ankunftszeit-Intervall und Abflugszeit-Intervall,
- der Ankunftsflughafen mit Ankunftszeit-Intervall und Abfahrtszeit-Intervall,
- der Ankunftsort mit Ankunftszeit-Intervall.

Die Zeitintervalle sind optional, es kann z. B. nur die Ankunftszeit am Ankunftsort oder die Abflugszeit am Flughafen angegeben werden. Wird keine Zeit eingetragen, wird die aktuelle Zeit als Abfahrtszeit genommen. Aus den angegebenen Wegpunkten werden nach folgenden Mustern Reiserouten gebildet, und zwar in dieser Reihenfolge:

- Abfahrtsort - Abflugshafen - Ankunftsflughafen [- Ankunftsort]
- [Abfahrtsort -] Abflugshafen - Ankunftsflughafen - Ankunftsort
- Abfahrtsort - Ankunftsort
- Abflugshafen - Ankunftsflughafen

Es ist möglich, mehrere Alternativen für einen Routenpunkt anzugeben, z. B. mehrere Abfahrtsorte bei einer Anfrage. Durch Kombination entstehen dann mehrere mögliche Reisenrouten, für die Verkehrsverbindungen zu ermitteln sind.

5.2.2 Die benötigten Arbeitsabläufe und Datenstrukturen

In diesem Abschnitt wird die Verfahrensweise bei der “Berechnung” von Reiseverbindungen erläutert. Dazu müssen die notwendigen Arbeitsabläufe sowie die Datenstrukturen für die Anfrage- und Produktionsdaten entworfen werden. In Abbildung 5.2 auf Seite 71 ist das ER-Modell der integrierten Anfragesicht dargestellt. Die Modellierung der Arbeitsabläufe erfolgt mit *Ereignisgesteuerten Prozeßketten*, siehe dazu auch Abschnitt 3.2.3.1. Prinzipiell läuft nachstehendes Verfahren zur Ermittlung von Verkehrsverbindungen zu einer Anfrage ab.

Im Arbeitsablauf *Anfrage stellen* (Abbildung 5.3 auf Seite 72) ist die Funktionsweise der Ausführung einer Anfrage dargestellt. Als erstes werden aus den Anfragedaten die möglichen Verkehrsrouten ermittelt und in der Datenbank abgespeichert. Jede Route besteht aus einer geordneten Liste von Teilrouten, die jeweils eine Wegstrecke zwischen zwei angegebenen Orten abdecken. In unserem Fall kann eine Route aus 1-3 Teilrouten bestehen, je nach Anfrage. Den Teilrouten werden die in der Anfrage angegebenen Ankunfts- und Abfahrtszeiten und die zur Berechnung zu benutzenden WWW-Datenquellen zugeordnet. Zwischen den Flughäfen wäre das Traveloverland, andernfalls Hafas. Danach wird für jede ermittelte Route der Arbeitsablauf *Route berechnen* aufgerufen. Wurde wenigstens eine Route erfolgreich abgeschlossen, wird die Ergebnispräsentation gestartet, die die gefundenen Verkehrsverbindungen zu den jeweiligen Reiserouten anzeigt.
Im Arbeitsablauf *Route berechnen* (Abbildung 5.4 auf Seite 73) werden als erstes

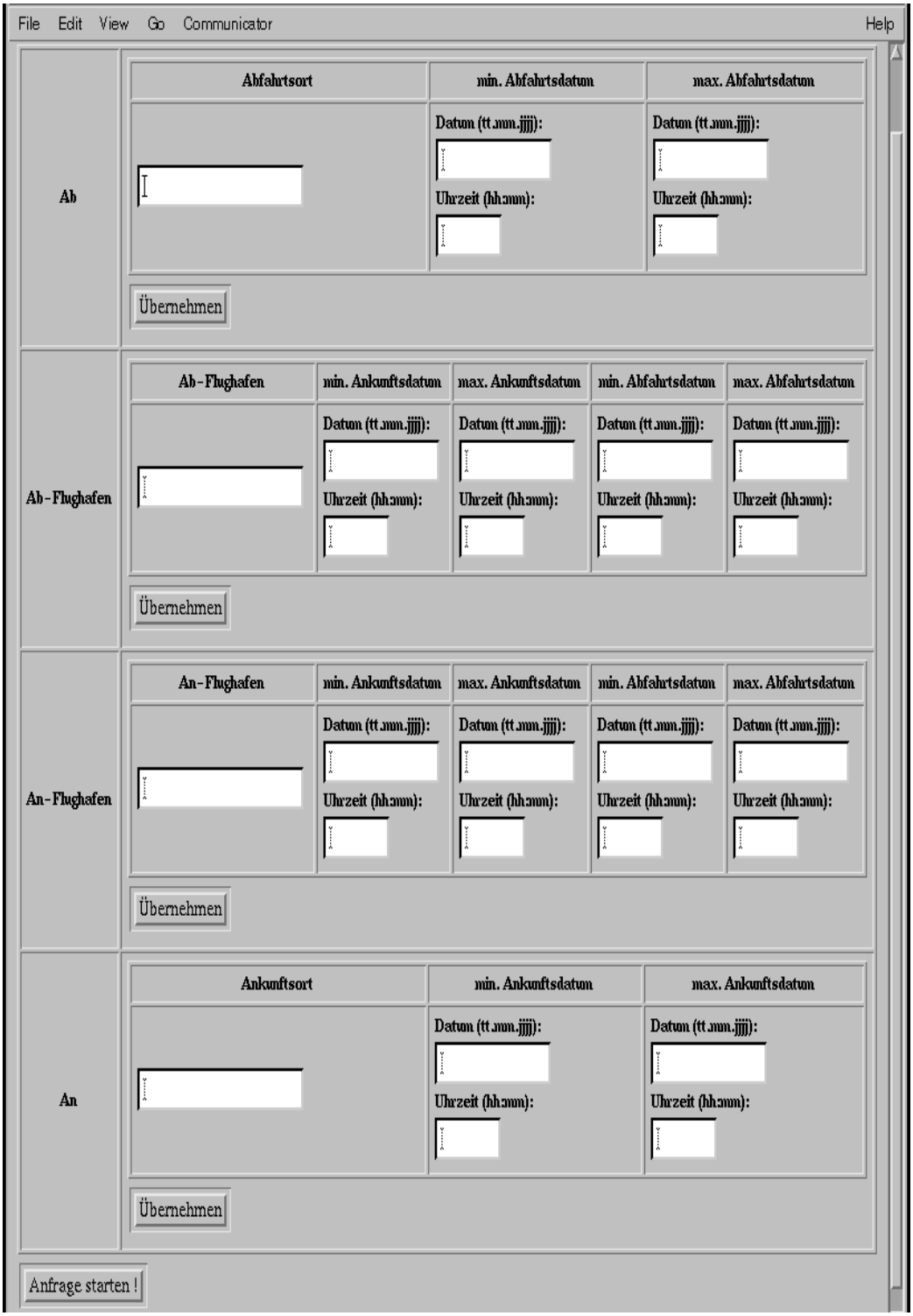

Abbildung 5.1: Anfrageschnittstelle der integrierten WWW-Anfragesicht *Reiseverbindungen*

die Teilrouten ermittelt, die mit ausreichend Daten für eine weitere Verarbeitung versorgt sind. Das sind die Teilrouten der Route, die eine Abfahrts- oder Ankunftszeit besitzen. Nun wird für jede ermittelte Teilroute der Arbeitsablauf *Teilroute berechnen* gestartet. Waren die Berechnungen erfolgreich, gibt es jetzt neue Abfahrts- oder Ankunftszeiten zu den Teilrouten. Daraus können die Abfahrts- oder Ankunftszeiten für noch nicht berechnete Vorgänger oder Nachfolger der fertigen Teilrouten ermittelt werden. Dann startet die Berechnung der Teilrouten erneut, bis zu allen Verkehrsverbindungen gefunden wurden. Scheitert die Berechnung einer Teilroute, scheitert die ganze Routenberechnung, da eine Teilstrecke fehlt.
Als nächstes erfolgt im Arbeitsablauf *Teilroute berechnen* (Abbildung 5.5 auf Seite 74) die Ermittlung der zugeordneten Datenquellen, die dann abgefragt werden. Liefert wenigstens eine WWW-Datenquelle ein Ergebnis, gibt es eine Verkehrsverbindung für diese Teilroute. Ansonsten ist die Teilroutenberechnung gescheitert.
Im Arbeitsablauf *Datenquelle abfragen* (Abbildung 5.6 auf Seite 75) wird der Wrapper für die abzufragende WWW-Datenquelle angebunden. Hier sind nur Hafas und Traveloverland berücksichtigt. Durch Erweiterung des Schemas können aber leicht weitere WWW-Datenquellen angeschlossen werden. Die Teilroute wird für die WWW-Datenquelle als fertig ausgetragen, um zu verhindern, daß im Arbeitablauf *Route berechnen* eine nochmalige Berechnung erfolgt.

Bei Traveloverland ist das Ergebnis über mehrere HTML-Seiten verteilt und muß so aus Daten, die von mehreren Wrappern stammen, zusammengesetzt werden. Dazu sind Kontroll- und Datenflußkonstrukte nötig, die als Programmcode der externen Applikation implementiert sind. Um die Funktionsweise eines zusammengesetzten Wrappers explizit zu beschreiben, bietet es sich an, ihn als Arbeitsablauf zu modellieren. Dadurch wird die Verständlichkeit erhöht und durch die erreichte Modularisierung kann die Ausführung der Teil-Wrapper parallelisiert werden. Abbildung 5.7 auf Seite 76 stellt einen alternativen Arbeitsablauf *Datenquelle abfragen* dar, wo statt der externen Applikation ein weiterer Arbeitsablauf *Traveloverland abfragen* aufgerufen wird.
Im Arbeitsablauf *Traveloverland abfragen* (Abbildung 5.8 auf Seite 77) werden zuerst die Fluggruppen zu einer Teilroute ermittelt. Danach wird für jede Gruppe der Arbeitsablauf *Traveloverland Flugverbindungen ermitteln* (Abbildung 5.9 auf Seite 78) gestartet. In diesem werden die zur Gruppe gehörenden Flugverbindungen abgefragt und jeweils der Arbeitsablauf *Traveloverland Flugdetails ermitteln* (Abbildung 5.10 auf Seite 79) aufgerufen.

In Anhang A.1 sind die aus den Arbeitsabläufen resultierenden Workflow-Schemata zu finden.

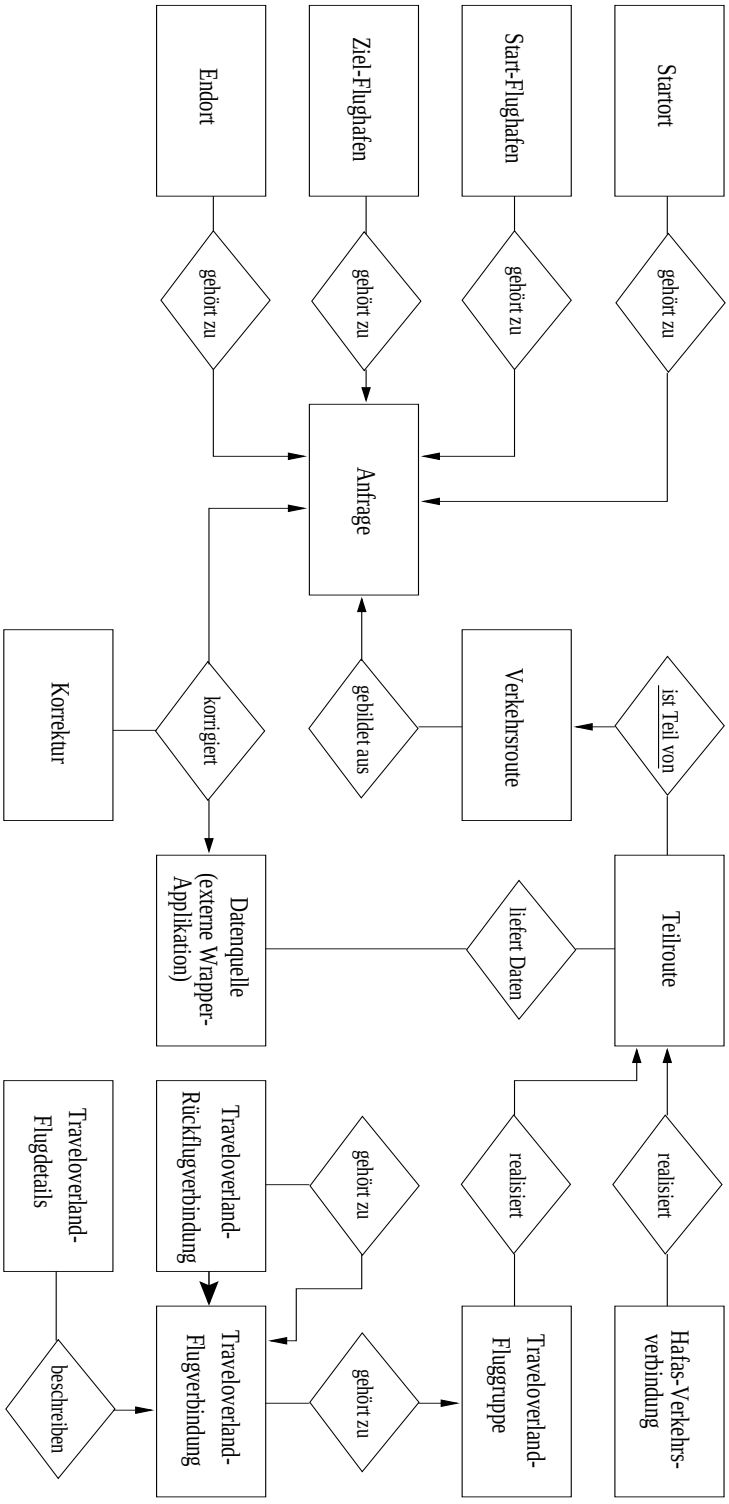

Abbildung 5.2: ER-Modell des Beispielszenarios *Reiseverbindungen*

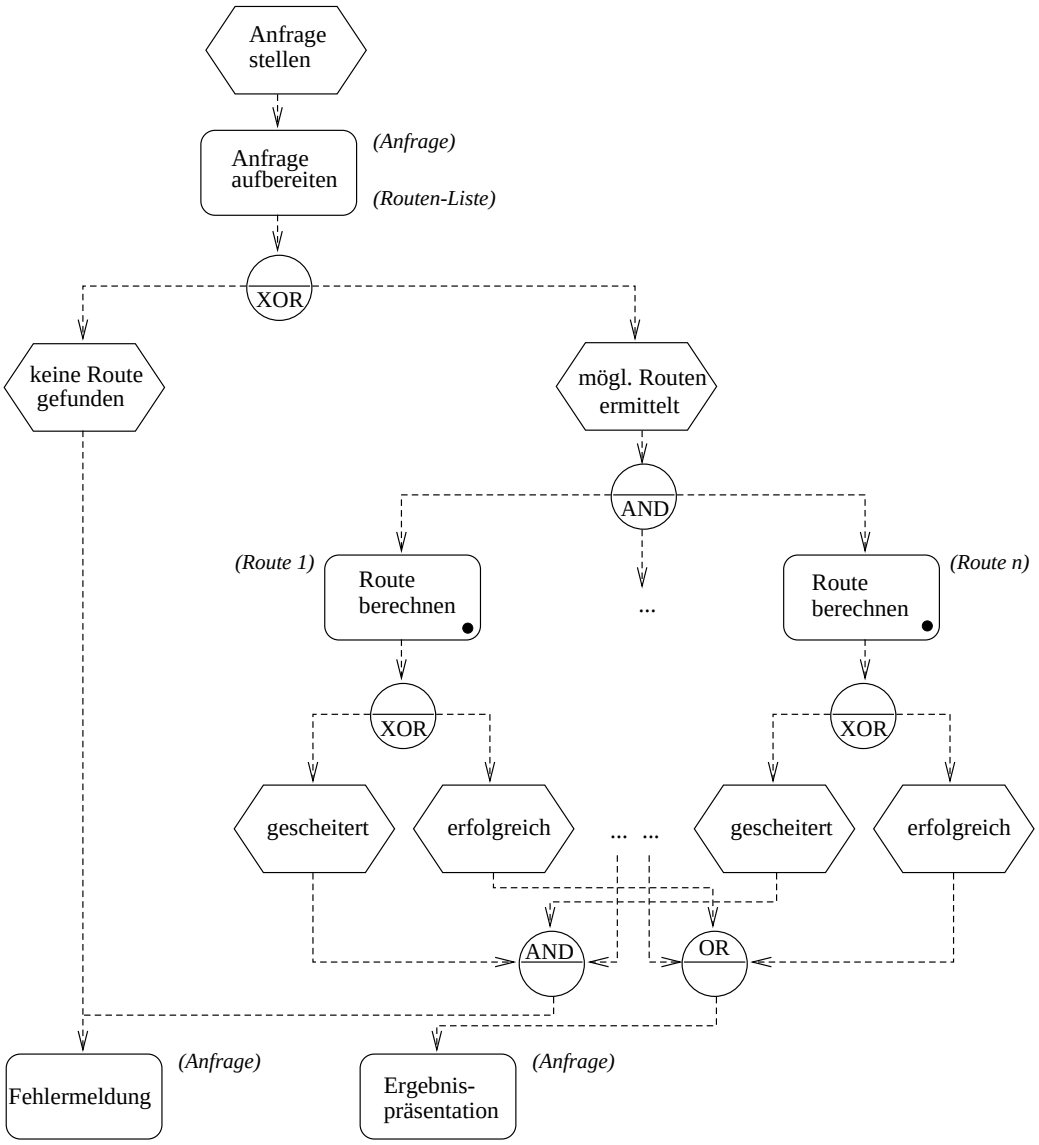

Abbildung 5.3: Arbeitsablauf *Anfrage stellen*

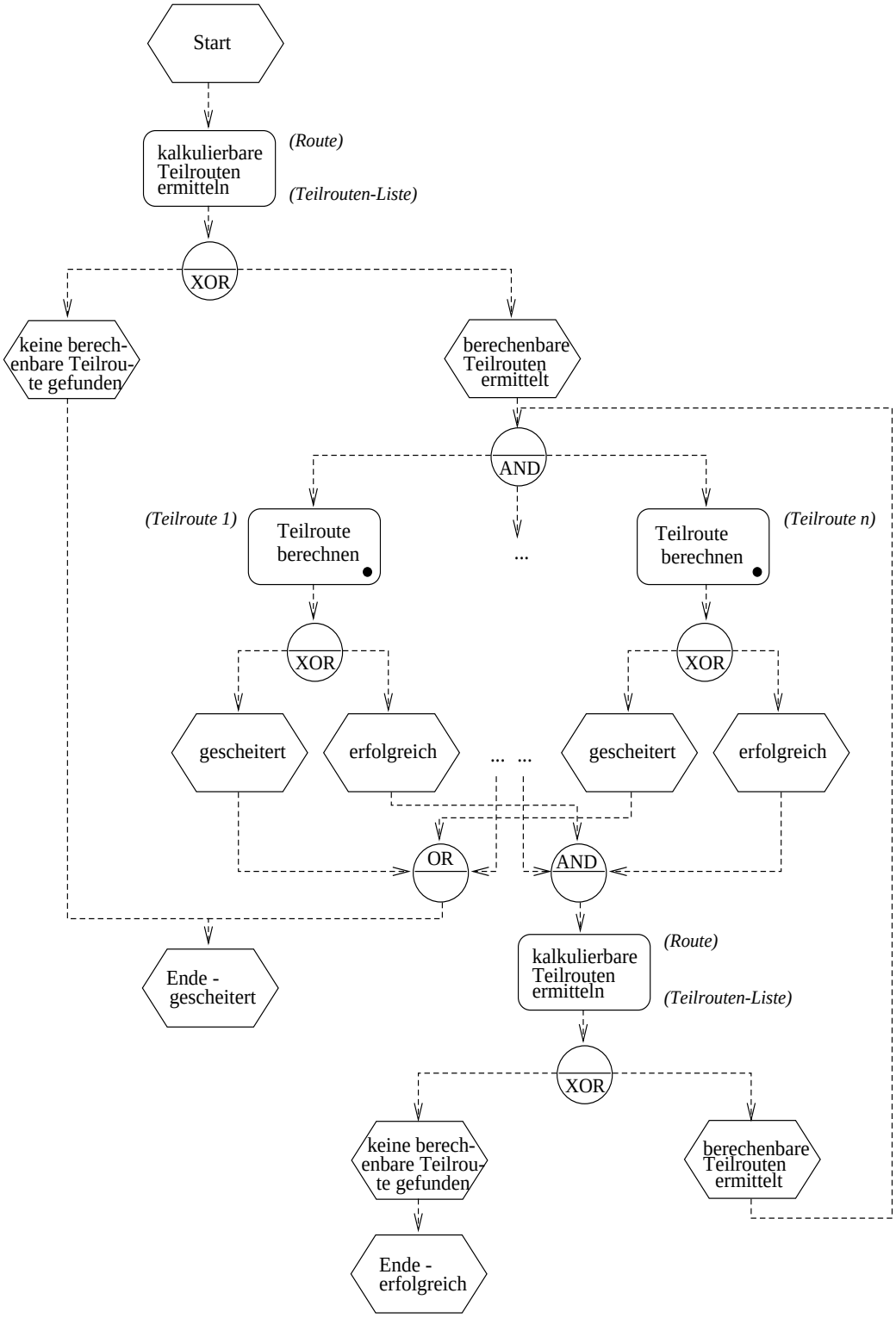

Abbildung 5.4: Arbeitsablauf *Route berechnen*

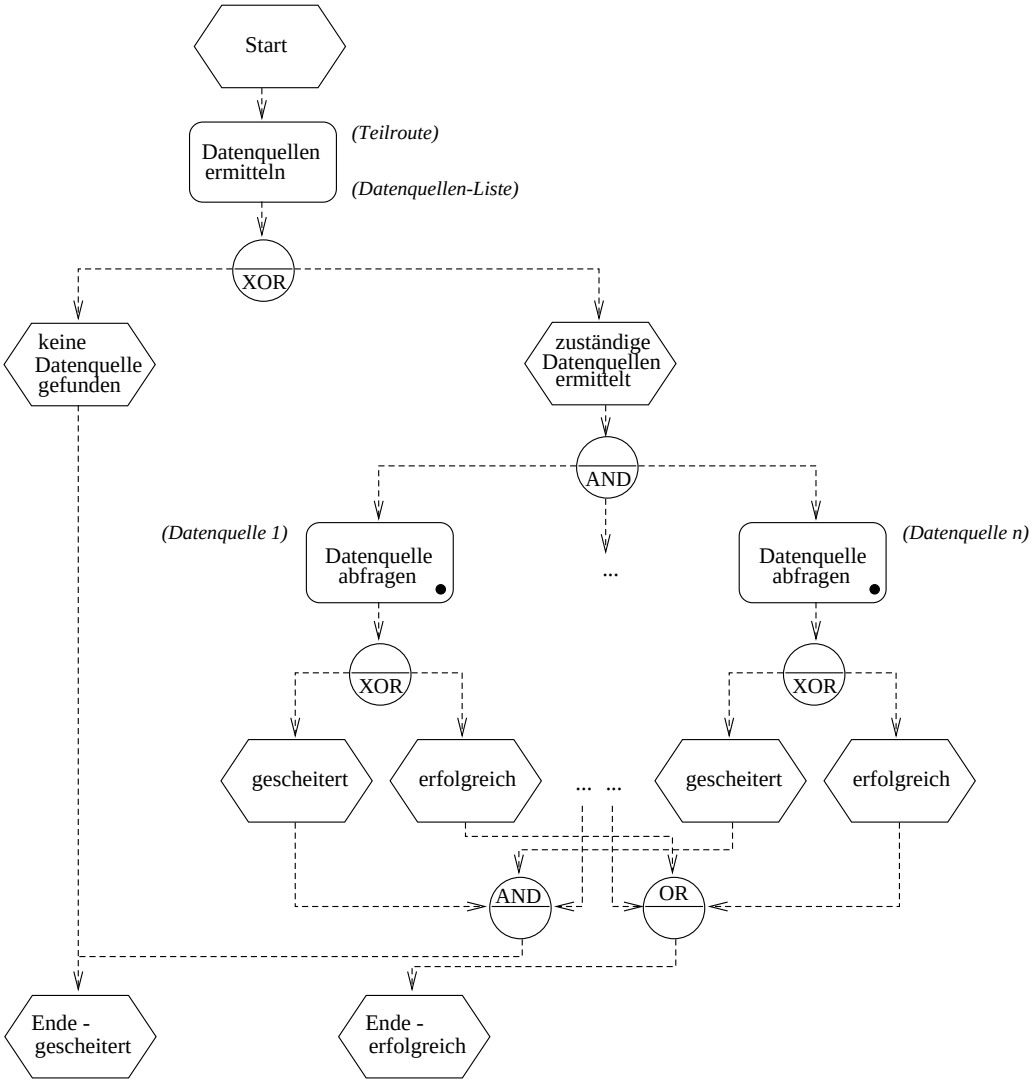

Abbildung 5.5: Arbeitsablauf *Teilroute berechnen*

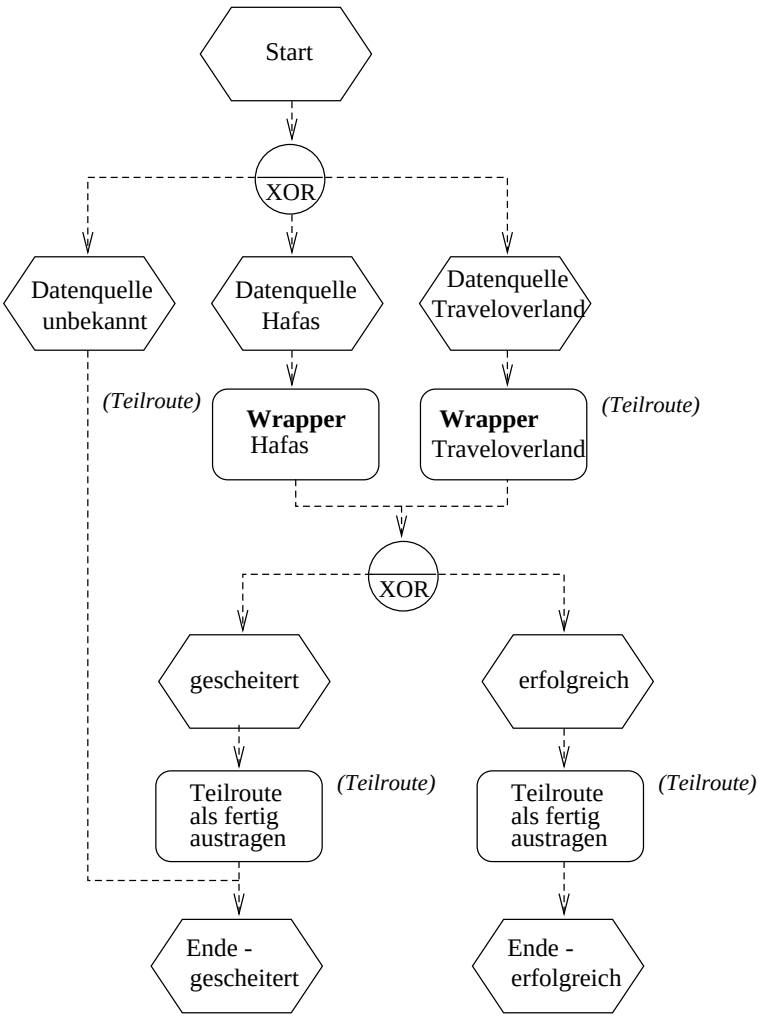

Abbildung 5.6: Arbeitsablauf *Datenquelle abfragen*

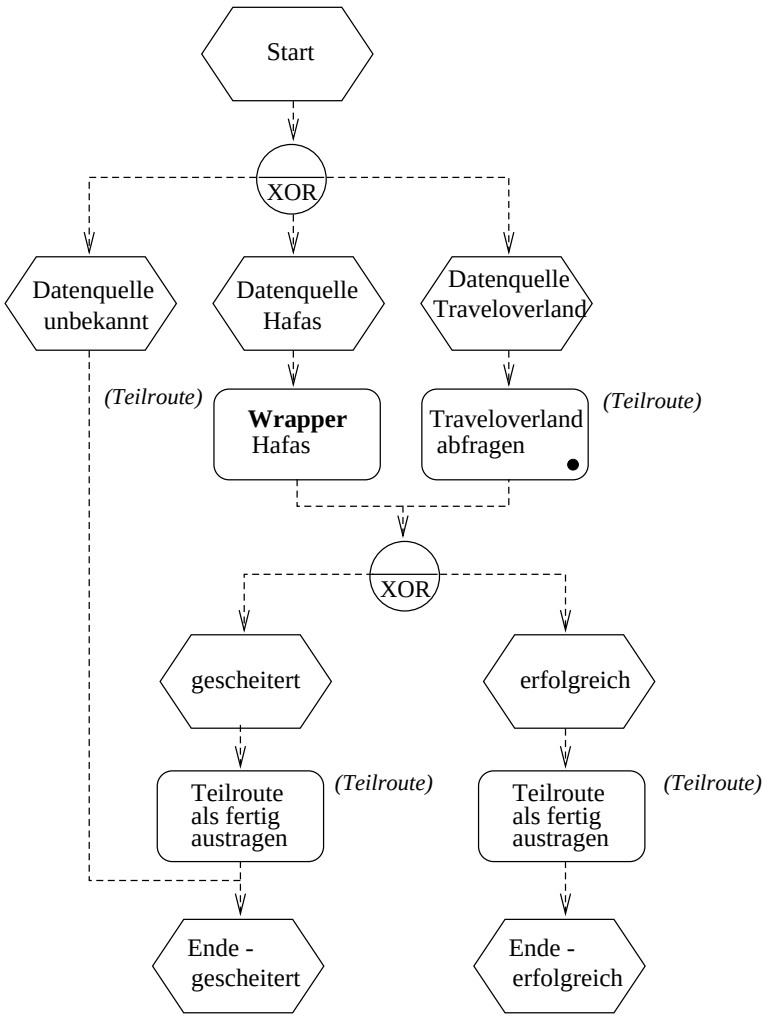

Abbildung 5.7: alternativer Arbeitsablauf *Datenquelle abfragen*

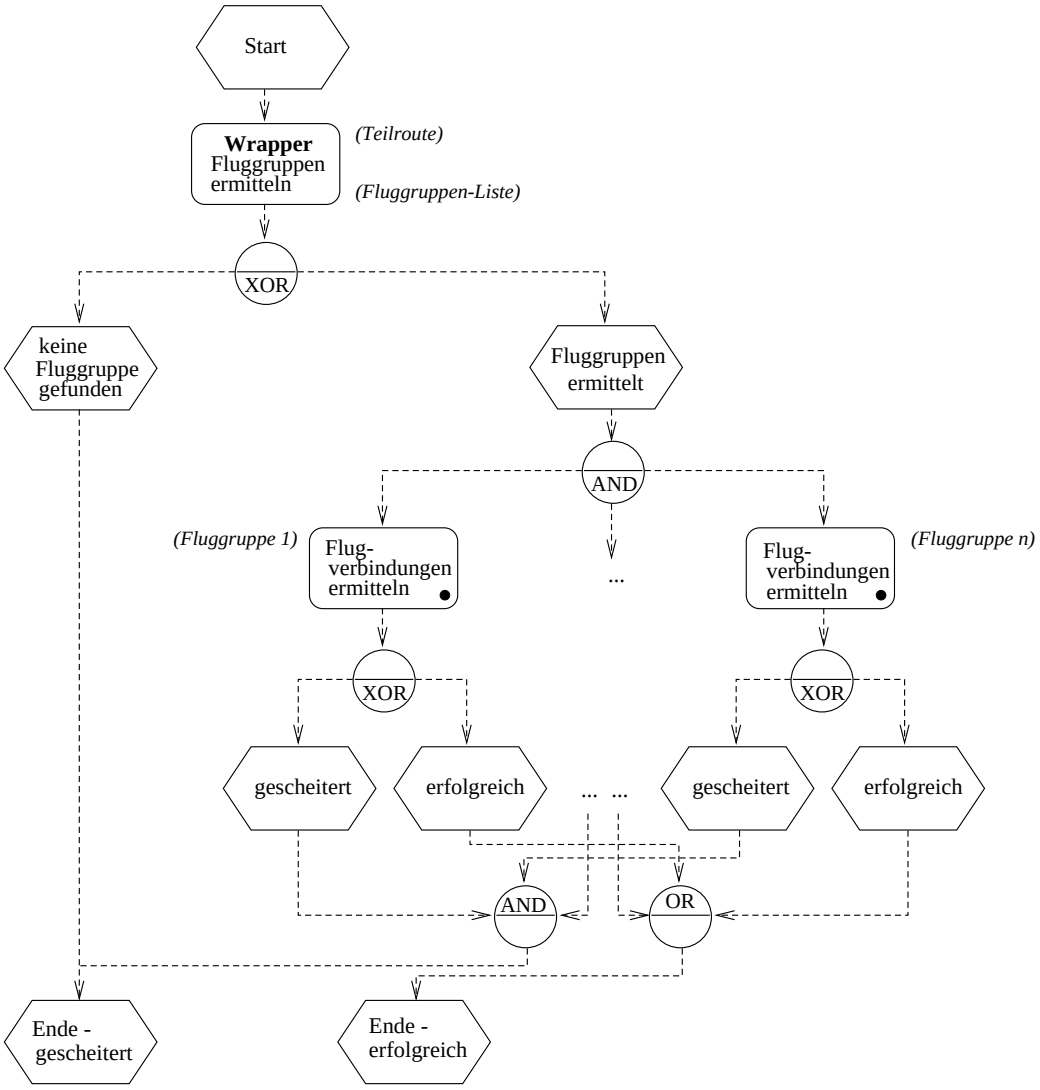

Abbildung 5.8: Arbeitsablauf *Traveloverland abfragen*

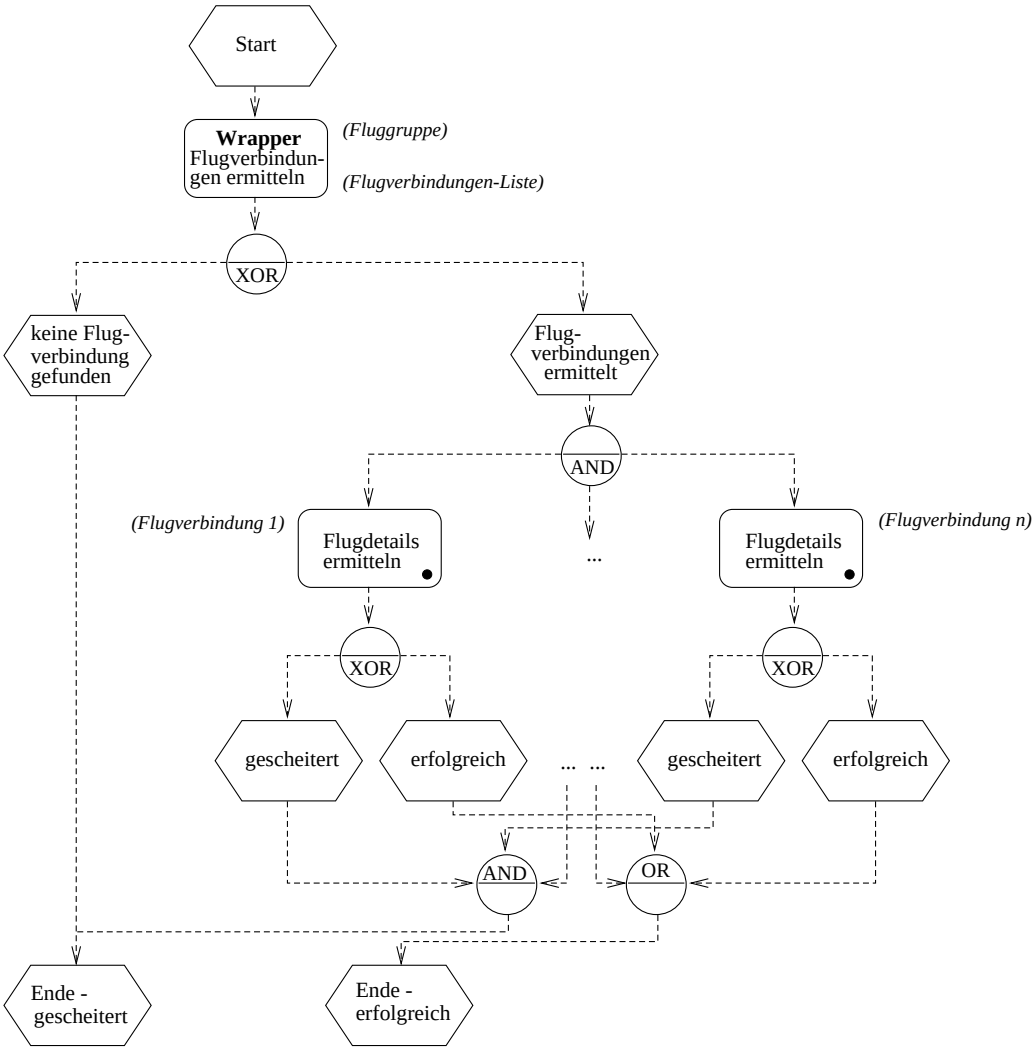

Abbildung 5.9: Arbeitsablauf *Traveloverland Flugverbindungen ermitteln*

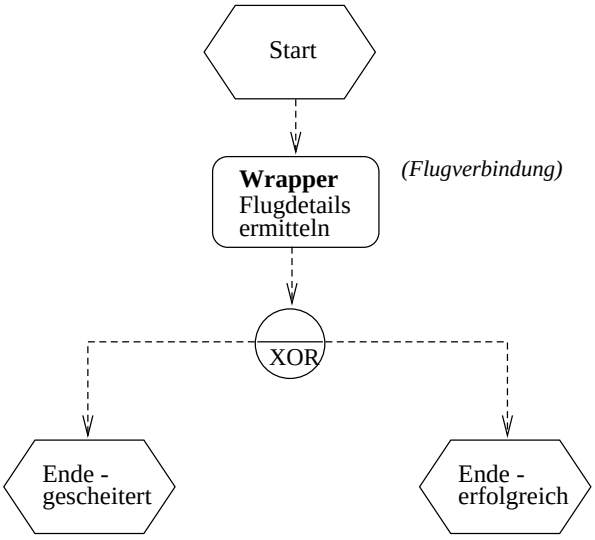

Abbildung 5.10: Arbeitsablauf *Traveloverland Flugdetails ermitteln*

5.2.3 Die Wrapper und externe Applikationen

Zur Abfrage der beiden WWW-Datenquellen Hafas und Traveloverland wurden Wrapper mit dem in Abschnitt 2.3.1 vorgestellten W4F-Toolkit implementiert. Der W4F-Compiler erzeugt aus der Wrapper-Spezifikation eine Java-Klasse, die in eine Java-Applikation eingebunden wurde. Der Wrapper bildet die extrahierten Daten auf dafür definierte Java-Klassen als Zieldatenstruktur ab. Auf diese greift die Java-Applikation zu und speichert die vom Wrapper gewonnenen Verkehrsverbindungsdaten in den entsprechenden Tabellen der Datenbank ab. Zum Zugriff auf das relationale DBMS OpenIngres wird die JDBC-Schnittstelle[2] verwendet. Die Java-Applikation ist über eine Workflow-Applikation an das WfMS angebunden. Diese startet die Java-Applikation mit den für die Wrapper-Steuerung notwendigen Anfrageparametern, empfängt Rückgabeparameter und erledigt die Kommunikation mit dem WfMS. Die Workflow-Applikationen sind wie das WfMS in der Programmiersprache Perl implementiert. In Anhang A.2 sind die benutzten Wrapper-Spezifikationen für die Datengewinnung abgebildet.

5.2.4 Die Interaktion mit dem Nutzer

Um die WWW-Anfragesicht praktikabel zu gestalten, muß der Nutzer die Möglichkeit haben, bestimmte auftretende Konflikte selbst aufzulösen. Der in diesem

[2]Eine Erläuterung von JDBC ist in [Dic97] zu finden.

Szenario am häufigsten auftretende Konflikt ist das Ablehnen der Ortsbezeichnungen durch die WWW-Datenquelle. Das muß der Wrapper erkennen und mögliche angegebene Alternativvorschläge extrahieren. Der Nutzer kann dann daraus eine Ortsbezeichnung auswählen oder eine neue Ortsbezeichnung eingeben. Die Wrapper-Applikation wird dann mit den korrigierten Daten neu gestartet. Nachstehend wird das Prinzip anhand der WWW-Datenquelle Hafas erläutert.
Die Wrapper-Applikation für die WWW-Datenquelle Hafas erkennt einen Konflikt mit dem Namen des Abfahrtsortes, den der Nutzer auflösen soll. Es wird eine Interaktionsinstanz mit einem eindeutigen Schlüssel angelegt, die dem Interaktionstyp *Abfahrtsort_mehrdeutig* zugeordnet ist. Notwendige Daten zur Interaktion werden ebenfalls abgespeichert, beispielsweise die Alternativvorschläge der Ortsbezeichnung. Die Wrapper-Applikation gibt als Rückgabeparameter den Interaktionsschlüssel an den Workflow zurück. Dieser fragt den Interaktionstyp der Interaktionsinstanz ab und startet die zugeordnete Interaktionsapplikation, hier ein CGI-Programm. Dieses erhält als Eingabeparameter den Interaktionsschlüssel und kann somit auf die Daten der Interaktion zugreifen. Es bietet dem Nutzer die Alternativvorschläge zur Auswahl an, oder läßt einen neue Ortsbezeichnung eingeben. Nach Beendigung des Nutzereingriffes wird ein Eintrag in der Korrekturtabelle angelegt, der neben dem beanstandeten Wert den neuen Wert enthält. Der Korrektureintrag ist einer Anfrage, für die er gilt, und einem Wrapper-Typ, hier Hafas, zugeordnet. Der Korrekturtyp des Eintrages bestimmt, für welchen Wrapper-Eingabeparameter die Korrektur gilt. Hier im Beispiel wäre das der Parameter *Abfahrtsort*. Die Interaktionsinstanz wird gelöscht und der Workflow startet die Workflow-Applikation, die die externe Wrapper-Applikation anbindet, mit denselben Eingabeparametern noch einmal. Diese sucht in der Korrekturtabelle nach Einträgen, welche auf die erhaltenen Parameter passen. Sie findet zum Wert des Parameters *Abfahrtsort* eine Korrektur und berichtigt den Parameter. Anschließend wird die externe Wrapper-Applikation mit den korrigierten Parametern aufgerufen.
Da der Korrektureintrag einem Wrapper-Typ zugeordnet ist, gilt er auch für andere Wrapper-Applikationen der Anfrage, die auf die Datenquelle Hafas zugreifen. So braucht der Nutzer nur einmal den Abfahrtsort per Hand korrigieren. Die Wrapper-Applikationen parallel ausgeführter Workflows benutzen dann automatisch den korrigierten Wert.

5.2.5 Probleme bei der Implementierung der integrierten WWW-Anfragesicht

Bei der Realisierung der Anfragesicht traten einige Probleme auf, viele kamen erst in der Implementierungsphase zum Vorschein. Nachstehend werden die we-

sentlichen Probleme aufgezählt.

- Die WWW-Datenquellen liefern nur zweistellige Jahresangaben. Diese müssen in ein vierstelliges Jahresformat konvertiert werden, um nicht einen Jahr 2000-Fehler zu produzieren.

- Die WWW-Datenquellen liefern zum Abfahrtsort das Abfahrtsdatum und die Abfahrtszeit, aber zum Ankunftsort wird nur die Ankunftszeit ausgegeben. Das korrekte Ankunftsdatum muß über die Reisedauer berechnet werden, da ja ein oder mehrere Tagwechsel stattgefunden haben können. Da die ermittelbaren Reiseverbindungen ganz Europa abdecken, müssen bei der Datumsberechnung die Zeitzonen berücksichtigt werden. Die WWW-Datenquellen liefern aber immer nur die lokale Ortszeit und keine Zeitzonenangaben. Daß aus der Reisedauer berechnete Ankunftsdatum kann sich durch eine mögliche Zeitverschiebung um einen Tag ändern. Durch Vergleich der berechneten mit der aus dem Fahrplan ermittelten Ankunftsstunde und Vergleich der zwischen beiden auftretenden Zeitdifferenz mit einem Schwellwert von 12 Stunden, ist das richtige Ankunftsdatum ermittelbar. (Größere Zeitzonen als +/-12 Stunden gibt es in Europa nicht.)

- Ein W4F-Wrapper weist einer Extraktionsvariable egal welchen Typs den Wert `null` zu, wenn die entsprechende Extraktionsregel keine Daten gefunden hat. Eigentlich müßte bei Abbildung auf einen Objekttyp dessen Konstruktor mit einer leeren NSL-Liste aufgerufen werden. Dieses Verhalten macht eine gesonderte Ausnahmebehandlung nötig, um nicht Methoden eines Objektes aufzurufen, das es nicht gibt.

- Wird statt des zusammengesetzten Wrappers für Traveloverland der alternative Workflow *Datenquelle abfragen* benutzt, erhöht sich die Systemlast des Rechners, auf dem das WfMS läuft, beträchtlich. Durch die Parallelisierung steigt die Anzahl der laufenden Prozesse stark an. Angenommen, es werden zu einer Anfrage 3 Fluggruppen ermittelt. Das Retrieval der zugehörigen Flugverbindungen (beispielsweise 5 pro Gruppe) erfolgt parallel in 3 Subworkflows. Für jede Flugverbindung wird wieder ein Subworkflow gestartet, der die Flugdetails ermittelt. Für jeden Wrapper-Aufruf wird ein Prozeß für die Workflow-Applikation zur Anbindung des externen Programmes und ein Prozeß für den Java-Interpreter, der das Wrapper-Programm ausführt, benötigt. Das ergibt 63 Prozesse. Außerdem muß man berücksichtigen, daß dazu noch weitere Reiserouten parallel berechnet werden können. Für einen hohen Parallelisierungsgrad ist das Prozeßkonzept nicht geeignet, da der Overhead einfach zu groß ist. Dafür müssen ande-

re Parallelisierungskonzepte, die nicht soviel Ressourcen brauchen, angewandt werden.

Kapitel 6

Zusammenfassung

Die drei vorgestellten Wrapper-Toolkits sind zur Implementierung von HTML-Wrappern gut geeignet. Konzeptionell hebt sich das Toolkit W4F von den anderen ab, da es ein neues Wrapper-Desgin einführt, um bestimmte vorteilhafte Wrapper-Eigenschaften (siehe Abschnitt 2.2) zu erreichen. Dabei unterscheidet es sich stark vom klassischen Programmiersprachen-Design, daß den beiden anderen Toolkits im wesentlichen zugrunde liegt. Allen gemeinsam ist aber die Einführung einer abstrakten HTML-Repräsentation, auf der die Sprachen der Toolkits arbeiten. Dadurch kann die Struktur des HTML-Dokumentes am besten zur Ermittlung der Struktur der zu extrahierenden Daten ausgenutzt werden.

Die Konzeption und Implementierung eines einfachen Workflow-Management-Systems ist ziemlich aufwendig. Zum Glück konnte auf das Workflow-Management-System aus der Studienarbeit [Goh99] als Basis zurückgegriffen werden, daß um benötigte Konstrukte erweitert wurde.

Die Realisierung der integrierten WWW-Anfragesicht *Reiseverbindungen* auf Basis eines WfMS hat sich als geeignet erwiesen. Durch die Modellierung der Arbeitsvorgänge werden die Arbeitsschritte sowie die benötigten Komponenten identifiziert und die Kontroll- und Datenflüsse zwischen ihnen explizit beschrieben. Die damit erreichte Modularisierung der Workflows sichert die Austauschbarkeit von Komponenten und eine gute Erweiterbarkeit der Anfragesicht. Besonders geeignet ist die Implementierung einer integrierten Anfragesicht mittels eines WfMS, wenn die aus den WWW-Datenquellen gewonnen Daten erst über komplexe Abläufe zu einem Gesamtergebnis durch andere Komponenten verarbeitet werden. Für kleine Extraktionsaufgaben ist die Nutzung eines WfMS aber überdimensioniert.
Die Integration der Auskunftssysteme der Deutschen Bahn und des Fluganbieters Traveloverland zu einer gemeinsamen Reiseverbindungsauskunft war prinzipiell

erfolgreich. Die Anfragesicht wurde so konzipiert, daß sie um andere Reiseauskunftssysteme erweitert werden kann. Probleme machen die Datenheterogenitäten der WWW-Datenquellen, beispielsweise bei den Ortsbezeichnungen, und die Lieferung ungenauer Daten, hier vor allem bei den Datumsangaben, die zur weiteren Berechnung der Reiseverbindung verwendet werden. Der Nutzer braucht auf jeden Fall Eingriffsmöglichkeiten, um auftretende Konflikte manuell auflösen zu können. Durch die völlige Autonomie der WWW-Datenquellen wird die Integrationslösung immer wieder angepaßt werden müssen.

Ausblick

Die Seitenbeschreibungssprache HTML ist momentan das Standardwerkzeug zur Repräsentation von Informationen im WWW. Entwickelt wurde sie zur Veröffentlichung von wissenschaftlichen Arbeiten. Mit dem rasanten Wachstum des WWW und dessen Einsatzmöglichkeiten stiegen aber genauso die Bedürfnisse nach mehr Darstellungsmöglichkeiten und der Interaktion mit dem Nutzer. Die Erweiterungen des HTML-Standards machen HTML zu einer universellen Seitenbeschreibungssprache zur Darstellung text- und bildbasierter Inhalte für einen menschlichen Nutzer. Von dem durch die technischen Weiterentwicklung ausgelösten Multimedia-Boom ist auch das WWW nicht ausgenommen. Dadurch wird es möglich, Menschen auch im Netz über ihre natürlichen Sinne anzusprechen und zu informieren, speziell in Bild und Ton. So ist auch verständlich, daß viele der sich im WWW präsentierenden Anbieter Multimedia-Elemente nutzen, um sich gegenüber Mitbewerbern abzusetzen. Diesem Bedarf tragen immer wieder neue Browser, die Javascript, ActiveX, Java beherrschen, sowie unterschiedlichste Browsererweiterungen für Grafik, Video, Musik usw. Rechnung. Durch die völlig normale Ausrichtung auf menschliche Bedürfnisse werden die Grenzen der automatisierten Datengewinnung über Wrapper aufgezeigt. Mit der Verdrängung der von Wrappern noch analysierbaren HTML-Sprache durch die genannten Elemente und dem Einsatz neuer an den Menschen angepaßter Darstellungs-Technologien wird der mögliche Einsatzbereich immer mehr reduziert. Um zukünftig Interoperabilität zwischen WWW-Datenquellen zu erreichen, muß über kurz oder lang eine Maschinen-Maschinen-Schnittstelle eingeführt werden.

Damit die Anfragesicht *Reiseverbindungen* praktisch einsetzbar ist, muß die Integration der WWW-Datenquellen verbessert werden. So sollte jedes angebundene Auskunftssystem die jeweils richtigen Ortsbezeichnung für die Übergangspunkte zwischen den Verkehrsnetzen erhalten, um Nutzerinteraktionen zu vermeiden. Man könnte beispielsweise zu jedem größeren Flughafen in Deutschland, den Haltestellennamen der Bahn- oder Busanbindung erfassen. Aus diesen Flughäfen hat

der Nutzer im Anfrageformular die Möglichkeit, seinen Zielflughafen zu wählen. Die so gestellte Anfrage trägt dann als Ankunftsort der Bahn- oder Busverbindung automatisch die richtige Haltestellenbezeichnung ein.
Eine weitere Verbesserung wäre die Schaffung zusätzlicher Interaktionsmöglichkeiten für den Nutzer, um sein Wissen zur Auflösung gefundener Konflikte zu nutzen. Schlägt eine Anfrage fehl, muß der Benutzer genauere Informationen über den Grund erhalten, um ihn bei einer Neuformulierung der Anfrage berücksichtigen zu können. Zum Beispiel könnte er die Fehlermeldungen der Auskunftssysteme angezeigt bekommen.
Abschließend kann man sagen, daß die Einbindung eines WWW-Anbieters in eine integrierte Anfragesicht wahrscheinlich nie die Qualität von dessen eigener Nutzerschnittstelle erreichen wird. Somit gehen Informationen verloren. Die Gesamtqualität einer integrierten Anfragesicht zu einem Einsatz-Szenario wird darüber entscheiden, ob diese Verluste gegenüber einem Komfortgewinn tolerierbar sind oder nicht.

Anhang A

Dokumentation zur integrierten Anfragesicht

A.1 Die Workflow-Schemata

A.1.1 Workflow-Schema *Anfrage stellen*

```
WORKFLOWTYPE Anfrage_stellen
  SUBWORKFLOWS
    Verbindung : Route_berechnen
  END_SUBWORKFLOWS
  # ---
  WORKFLOW_VARIABLES
    IN: anfrage_key: string
        routen : list(string)
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION Ergebnisausgabe
      IN anfrage_key
      OPERATIONBODY
        HTTP_REDIRECT: ergebnisausgabe.cgi
      END_OPERATIONBODY
    END_OPERATION
    # ---
    OPERATION Anfrage_aufbereiten
      IN anfrage_key
```

```
      OUT routen
      OPERATIONBODY
        RUN_FILE: routen_ermitteln.pl
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    START_OP Anfrage_aufbereiten;
    IF (Anfrage_aufbereiten) THEN {
       START_WF[Verbindung(anfrage_key, route)
                FOREACH route FROM routen SUCCESS BY ANY];
       IF (Verbindung) THEN {
          START_OP Ergebnisausgabe;
       }
       ENDIF;
    }
    ENDIF;
    RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
    INITIAL_EVENT: ANFRAGE_BERECHNEN
    PUBLIC_NAME: Reiseverbindungsanfrage stellen
    DESCRIPTION
      Ermitteln von Reiseverbindungen zu einer Anfrage
    END_DESCRIPTION
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.2 Workflow-Schema *Route berechnen*

```
WORKFLOWTYPE Route_berechnen
  SUBWORKFLOWS
    Teilverbindung: Teilroute_berechnen
  END_SUBWORKFLOWS
  # ---
  WORKFLOW_VARIABLES
    IN: route: string
        teilrouten: list(string)
        erfolgs_flag: boolean
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION ermittle_teilrouten
      IN route
      OUT teilrouten
      OPERATIONBODY
        RUN_FILE: teilrouten_aufbereiten.pl
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    START_OP ermittle_teilrouten;
    IF (ermittle_teilrouten) THEN {
       SET erfolgs_flag (1);
    }
    ELSE {
       SET erfolgs_flag (0);
    }
    ENDIF;
    WHILE (ermittle_teilrouten) {
      START_WF[Teilverbindung(route, teilroute)
               FOREACH teilroute
               FROM teilrouten SUCCESS BY ALL];
      SET erfolgs_flag CALCULATE((erfolgs_flag) &&
                                 (Teilverbindung));
      START_OP ermittle_teilrouten;
    }
    ENDWHILE;
```

```
    RETURN WITH STATUS (erfolgs_flag);
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
     ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.3 Workflow-Schema *Teilroute berechnen*

```
WORKFLOWTYPE Teilroute_berechnen
  SUBWORKFLOWS
    Verkehrsverbindung: Datenquelle_abfragen
  END_SUBWORKFLOWS
  # ---
  WORKFLOW_VARIABLES
    IN: teilroute: string
    IN: route: string
        wrapper: list(string)
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION ermittle_wrapper
      IN teilroute, route
      OUT wrapper
      OPERATIONBODY
        RUN_FILE: ermittle_wrapper.pl
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    START_OP ermittle_wrapper;
    START_WF[Verkehrsverbindung(route,teilroute,datenquelle)
             FOREACH datenquelle FROM wrapper SUCCESS BY ANY];
    RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.4 Workflow-Schema *Datenquelle abfragen*

```
WORKFLOWTYPE Datenquelle_abfragen
  WORKFLOW_CONSTANTS
    hafas: string (HAFAS)
    traveloverland: string (TRAVELOVERLAND)
  END_WORKFLOW_CONSTANTS
  # ---
  WORKFLOW_VARIABLES
    IN: datenquelle: string
    IN: teilroute: string
    IN: route: string
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION verbindung_hafas OF OPERATIONTYPE wrapper_hafas
      IN route, teilroute
      OPERATIONBODY
      END_OPERATIONBODY
    END_OPERATION
    # ---
    OPERATION verbindung_traveloverland
    OF OPERATIONTYPE wrapper_traveloverland
      IN route, teilroute
      OPERATIONBODY
      END_OPERATIONBODY
    END_OPERATION
    # ---
    OPERATION update_teilroute
      OF OPERATIONTYPE teilroutenberechnung_wrapper_abschliessen
      IN route, teilroute, datenquelle
      OPERATIONBODY
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    IF (datenquelle eq hafas) THEN {
       START_OP verbindung_hafas;
       START_OP update_teilroute;
    }
```

```
    ELSE {
       IF (datenquelle eq traveloverland) THEN {
          START_OP verbindung_traveloverland;
          START_OP update_teilroute;
       }
       ENDIF;
    }
    ENDIF;
  RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.5 Alternatives Workflow-Schema *Datenquelle abfragen*

```
WORKFLOWTYPE Datenquelle_abfragen
  SUBWORKFLOWS
    Flug_Traveloverland: Traveloverland_abfragen
  END_SUBWORKFLOWS
  # ---
  WORKFLOW_CONSTANTS
    hafas: string (HAFAS)
    traveloverland: string (TRAVELOVERLAND)
  END_WORKFLOW_CONSTANTS
  # ---
  WORKFLOW_VARIABLES
    IN: datenquelle: string
    IN: teilroute: string
    IN: route: string
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION verbindung_hafas OF OPERATIONTYPE wrapper_hafas
      IN route, teilroute
      OPERATIONBODY
      END_OPERATIONBODY
    END_OPERATION
    # ---
    OPERATION update_teilroute
      OF OPERATIONTYPE teilroutenberechnung_wrapper_abschliessen
      IN route, teilroute, datenquelle
      OPERATIONBODY
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    IF (datenquelle eq hafas) THEN {
       START_OP verbindung_hafas;
       START_OP update_teilroute;
    }
    ELSE {
       IF (datenquelle eq traveloverland) THEN {
          START_WF[Flug_Traveloverland(route, teilroute)];
```

```
          START_OP update_teilroute;
        }
        ENDIF;
    }
    ENDIF;
  RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.6 Workflow-Schema *Traveloverland abfragen*

```
WORKFLOWTYPE Traveloverland_abfragen
  SUBWORKFLOWS
    Verbindungen: Traveloverland_Flugverbindungen_ermitteln
  END_SUBWORKFLOWS
  # ---
  WORKFLOW_VARIABLES
    IN: teilroute: string
    IN: route: string
        fluggruppen: list(string)
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION hole_fluggruppen
      IN route, teilroute
      OUT fluggruppen
      OPERATIONBODY
        RUN_FILE: wrapper_TO_fluggruppen.pl
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    START_OP hole_fluggruppen;
    IF (hole_fluggruppen) THEN {
       START_WF[Verbindungen(fluggruppe)
                FOREACH fluggruppe
                FROM fluggruppen SUCCESS BY ANY];
    }
    ENDIF;
    RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.7 Workflow-Schema *Traveloverland Flugverbindungen ermitteln*

```
WORKFLOWTYPE Traveloverland_Flugverbindungen_ermitteln
  SUBWORKFLOWS
    Fluege: Traveloverland_Flugdetails_ermitteln
  END_SUBWORKFLOWS
  # ---
  WORKFLOW_VARIABLES
    IN: fluggruppe: string
        flugverbindungen: list(string)
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION hole_flugverbindungen
      IN fluggruppe
      OUT flugverbindungen
      OPERATIONBODY
        RUN_FILE: wrapper_TO_flugverbindungen.pl
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    START_OP hole_flugverbindungen;
    IF (hole_flugverbindungen) THEN {
       START_WF[Fluege(flugverbindung)
                FOREACH flugverbindung
                FROM flugverbindungen SUCCESS BY ANY];
    }
    ENDIF;
    RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.1.8 Workflow-Schema *Traveloverland Flugdetails ermitteln*

```
WORKFLOWTYPE Traveloverland_Flugdetails_ermitteln
  WORKFLOW_VARIABLES
    IN: flugverbindung: string
  END_WORKFLOW_VARIABLES
  # ---
  OPERATIONS
    OPERATION hole_flugdetails
      IN flugverbindung
      OPERATIONBODY
        RUN_FILE: wrapper_TO_flugdetails.pl
      END_OPERATIONBODY
    END_OPERATION
  END_OPERATIONS
  # ---
  CONTROL_FLOW
    START_OP hole_flugdetails;
    RETURN;
  END_CONTROL_FLOW
  # ---
  ORGANIZATIONAL_POLICY
    ROLE: Anfrager
  END_ORGANIZATIONAL_POLICY
END_WORKFLOWTYPE
```

A.2 Die W4F-Wrapper

A.2.1 Spezifikation des Hafas-Wrappers

```
SCHEMA {
  String link_spaeter;
  String link_frueher;
   Status ab_status;
  Status an_status;
  String[] ab_orte;
  String[] an_orte;
  Verbindung_Hafas[] verbindungen;
}
EXTRACTION_RULES {
  verbindungen=html.body.form[0].table[0].tr[i:*]
    (.td[1]->pcdata[0].txt // Ab Hbf
    #.td[1]->pcdata[1].txt // An Hbf
    #.td[2]->pcdata[0].txt // Datum
    #.td[4]->pcdata[0].txt // Zeit ab
    #.td[4]->pcdata[1].txt // Zeit an
    #.td[5]->pcdata[0].txt // Umsteiganzahl
    #.td[6]->pcdata[0].txt // Dauer
    #.td[7]->pcdata[0].txt, split /, / // Zugtypen
    #.td[8]->pcdata[0].txt,
        match /([0-9]+,([0-9][0-9])?)/ // Preis
    )
    WHERE html.body.form[0].table[0].tr[i].td[3].txt=~ "ab";
  link_frueher=html->a[j].getAttr(href)
    WHERE html->a[j]->pcdata[0].txt =~ "frühere Verbindungen";
  link_spaeter=html->a[j].getAttr(href)
    WHERE html->a[j]->pcdata[0].txt =~ "spätere Verbindungen";
  ab_status=html->pcdata[i].txt WHERE html->pcdata[i].txt =~
    "Starthaltestelle ist nicht eindeutig.";
  an_status=html->pcdata[i].txt WHERE html->pcdata[i].txt =~
    "Zielhaltestelle ist nicht eindeutig.";
  ab_orte=html->tr[i]->select[0].option[*].pcdata[0].txt
    WHERE html->tr[i]->pcdata[0].txt =~ "von:";
  an_orte=html->tr[i]->option[*].pcdata[0].txt
    WHERE html->tr[i]->pcdata[0].txt =~ "nach:";
}
```

```
RETRIEVAL_RULES {
  get(String url, String ab, String an, String datum,
      String zeit, String timesel)
    {
      METHOD: POST;
      URL: "$url$";
      PARAM: "from"=$ab$,
             "to"=$an$,
             "time"=$zeit$,
             "timesel"=$timesel$,
             "start"="Verbindungen suchen",
             "filter"="ALL#0#0#0",
             "via.1"="",
             "date"=$datum$,
             "protocol"="http:";
    }
  get_more(String url)
    {
      METHOD: GET;
      URL: "$url$";
    }
}
```

HTML-Beispiel

Der nachstehende HTML-Quelltext zeigt einen Verbindungseintrag aus der Ergebnistabelle für eine Anfrage.

```
...
<TR VALIGN="TOP">
<TD VALIGN="MIDDLE"> <INPUT TYPE=CHECKBOX
CHECKED NAME="sel2"></TD>
<TD NOWRAP COLSPAN="2">
<FONT SIZE="2"><B><A HREF="/bin/bhftafel.exe/ds/8010304/23:03">Rostock
Hbf</A>
<br><A HREF="/bin/bhftafel.exe/ds/8002461/9:14/an">G&uuml;tersloh
Hbf</A></B></FONT></TD>
<TD ALIGN=LEFT NOWRAP>
<FONT SIZE="2">
<B> 15.11.99 </B>
</FONT>
```

```
</TD>
<INPUT TYPE=HIDDEN NAME=date VALUE="15.11.99">
<TD ALIGN=LEFT NOWRAP><FONT SIZE="2"><B>ab<BR>an</B></FONT></TD>
<TD ALIGN=LEFT NOWRAP>
<FONT SIZE="2">
<B> 23:03 <BR> 9:14 </B>
</FONT>
</TD>
<TD ALIGN=LEFT NOWRAP><FONT SIZE="2"><B>4</B></FONT></TD>
<TD ALIGN=LEFT NOWRAP>
<FONT SIZE="2">
<B> 10:11 </B>
</FONT>
</TD>
<TD ALIGN=LEFT><FONT SIZE="2"><B>
<A HREF="http://bahn.hafas.de/zugart_ds.html" TARGET="_BLANK">RE</A>,
<A HREF="http://bahn.hafas.de/zugart_ds.html" TARGET="_BLANK">ICE</A>
</B></FONT></TD>
<TD ALIGN=LEFT NOWRAP COLSPAN="2">
<FONT SIZE="2">
<B>156,00 DM<BR>Standardtarif</B>
</FONT>
</TD>
<TR VALIGN="TOP">
...
```

A.2.2 Die Wrapper für die Traveloverland-Seiten

Spezifikation des Wrappers *Fluggruppen*

```
SCHEMA {
  Fluggruppe_TO[] fluggruppen;
  Status ab_status;
  Status an_status;
  Status ab_datum_status;
  String[] abflug_nur_ab;
  String args_value;
}
EXTRACTION_RULES {
  fluggruppen= html.body->form[0].table[*]
    (.tr[0]->input[0].getAttr(value)
    #.tr[4]->a[0].getAttr(href) );
  ab_status=html->pcdata[i].txt
    WHERE html->pcdata[i].txt =~ "Unbekannter Abflugort";
  an_status=html->pcdata[i].txt
    WHERE html->pcdata[i].txt =~ "Unbekannter Zielort";
  ab_datum_status=html->pcdata[i].txt WHERE
    html->pcdata[i].txt =~"Abflugdatum liegt vor heutigem Datum";
  abflug_nur_ab= html->table[i]->ul[0]->li[*].pcdata[0].txt
    WHERE html->table[i]->tr[0].td[1].pcdata[0].txt =~
    "nur ab den folgenden Flugh";
  args_value= html->input[i].getAttr(value)
  WHERE html->input[i].getAttr(name)="args";
}
RETRIEVAL_RULES {
  get(String url, String ab, String an, String ab_datum,
      String rueck_datum, String anzahl, String flugart,
      String fluggesellschaft, String klasse, String jugendtarif)
    {
      METHOD: POST;
      URL: "$url$";
      PARAM: "dep"=$ab$,
             "dest"=$an$,
             "date1"=$ab_datum$,
             "date2"=$rueck_datum$,
             "npax"=$anzahl$,
             "owrt"=$flugart$,
```

```
            "airline"=$fluggesellschaft$,
            "class"=$klasse$,
            "ys"=$jugendtarif$;
    }
}
```

HTML-Beispiel

Der nachstehende HTML-Quelltext zeigt eine Fluggruppe der ersten Ergebnisseite einer Anfrage.

```
...
<TABLE BORDER=0 WIDTH=470>
<TR VALIGN=TOP BGCOLOR="#f0f0f0">
<TD ROWSPAN=10 WIDTH=15>
<INPUT TYPE="checkbox" NAME="flight" VALUE="MUC|NYC|44|61.50">
</TD><TD COLSPAN=2><STRONG>DM 426</STRONG>
<TR><TD COLSPAN=2>zuzüglich Flughafensteuern DM 61.50
= <strong>DM 487.50</strong><BR>
</TD></TR>
<TR><TD>mit: US Airways <BR></TD>
<TD ROWSPAN=5 ALIGN=RIGHT>
<INPUT TYPE="submit" VALUE="Verfügbarkeit"
NAME="MUC|NYC|44|61.50">
</TD></TR>
<TR><TD>Route:
<A HREF="http://www.munich-airport.de/">MUC</A>
- NYC via: Philadelphia<BR>
</TD></TR>
<TR><TD>
<A HREF="/cgi/fd?MUC|NYC|44|61.50|1|4|2451526|">Tarifdetails</A>
</TD></TR>
</TABLE>
...
```

Spezifikation des Wrappers *Flugverbindungen*

```
SCHEMA {
  Status kein_rueckflug;
  String args_value;
  Flugverbindung_TO[] verbindungen;
}
EXTRACTION_RULES {
  kein_rueckflug=html->input[i].getAttr(value)
    WHERE html->input[i].getAttr(value)=~"Rückflügen";
  args_value=html->input[i].getAttr(value)
    WHERE html->input[i].getAttr(name)="args";
  verbindungen=html.body[0].table[1].tr[0].td[1].form[0].table[0].tr[2-]
    (.td[1].table[0].tr[1-].td[1].a[0].getAttr(href)
    #.td[0].input[0].getAttr(value)
    );
}
RETRIEVAL_RULES {
  get(String url, String flug_key, String args_value)
    {
      METHOD: POST;
      URL: "$url$";
      PARAM: "flight"=$flug_key$,
             "args"=$args_value$;
    }
}
```

HTML-Beispiel

Der nachstehende HTML-Quelltext zeigt eine Flugverbindung aus der Tabelle mit Flugverbindungen zu einer Fluggruppe.

```
...
<TR ALIGN=CENTER><TD>23.12.99</TD>
<TD><A HREF="/cgi/vi?LH45:2451536">LH 45</A></TD>
<TD>14:55</TD>
<TD>16:05</TD>
<TD>Hamburg HAM</TD>
<TD>Frankfurt FRA</TD></TR>
...
```

Spezifikation des Wrappers *Flugdetails*

```
SCHEMA {
  String flugnr;
  String ab_datum;
  String ab;
  String an;
  String ab_zeit;
  String an_zeit;
  String dauer;
  String meilen;
  String fluggeraet;
  String fh_ab_info;
  String fh_an_info;
}
EXTRACTION_RULES {
  flugnr=html->pcdata[i].txt, match /[ ]+(.+)\,/
    WHERE html->pcdata[i].txt=~"Flugnummer";
  ab_datum=html->pcdata[i].txt,
    match /([0-9]+\.[0-9]+\.[0-9]+)/
    WHERE html->pcdata[i].txt=~"Flugnummer";
  ab=html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
          pcdata[0].txt="von";
  an=html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
          pcdata[0].txt="nach";
  ab_zeit=html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
          pcdata[0].txt="Abflug";
  an_zeit= html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
          pcdata[0].txt="Ankunft";
  dauer= html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
          pcdata[0].txt="Dauer";
  meilen= html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
          pcdata[0].txt="Meilen";
  fluggeraet= html->table[1]->table[0].tr[1].td[j].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[0].th[j].
```

```
          pcdata[0].txt="Fluggerät";
  fh_ab_info= html->table[1]->table[0].tr[i].td[0].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[i].td[0].
          pcdata[0].txt=~"Abflug";
  fh_an_info= html->table[1]->table[0].tr[i].td[0].pcdata[0].txt
    WHERE html->table[1]->table[0].tr[i].td[0].
     pcdata[0].txt=~"Ankunft";
}
RETRIEVAL_RULES {
  get(String url)
    {
      METHOD: GET;
      URL: "$url$";
    }
}
```

HTML-Beispiel

Der nachstehende HTML-Quelltext zeigt die Details zu einer Flugverbindung.

```
...
<H2>Fluginformationen</H2>
<STRONG>Flugnummer LH7, 23.12.1999</STRONG><P>
<TABLE BORDER=0 BGCOLOR="#ffcccc">
<TR><TH>von</TH>
<TH>nach</TH>
<TH>Abflug</TH>
<TH>Ankunft</TH>
<TH>Dauer</TH>
<TH>Meilen</TH>
<TH>Fluggerät</TH></TR>
<TR ALIGN=CENTER><TD>Hamburg HAM</TD>
<TD>Frankfurt FRA</TD>
<TD>06:40</TD>
<TD>07:50</TD>
<TD> 1:10</TD>
<TD>257</TD>
<TD>Airbus A321</TD></TR>
<TR ALIGN=RIGHT><TD COLSPAN=7>Abflug HAM: TERMINAL 4</TD></TR>
<TR ALIGN=RIGHT><TD COLSPAN=7>Ankunft FRA: TERMINAL 1</TD></TR>
</TABLE>
...
```

A.3 Die Datenbankrelationen

Die Daten des Beispielszenarios *Reiseverbindungen* werden in der OpenIngres-Datenbank `verbindung` gespeichert. Folgende Relationen nehmen die Anfrage-, Zwischen- und Ergebnisdaten auf:

- `ANFRAGE`
  Speichert die Daten zu einer Reiseverbindungsanfrage.
- `START_ORT`
  Enthält die Abfahrtsorte der Verbindungsanfragen.
- `END_ORT`
  Enthält die Ankunftsorte der Verbindungsanfragen.
- `VIA_AB`
  Enthält die Abflugshäfen der Verbindungsanfragen.
- `VIA_AN`
  Enthält die Ankunftsflughäfen der Verbindungsanfragen.
- `KORREKTUR`
  Speichert die Korrekturwerte zu einer Anfrage.
- `ROUTE`
  Enthält die zu einer Anfrage generierten Reiserouten.
- `TEILROUTE`
  Speichert die Teilrouten einer Reiseroute.
- `TEILRT_WRAPPER`
  Speichert, welche Teilroute von welchem Wrapper-Typ berechnet wird.
- `WRAPPER`
  Speichert die Wrapper-Typen, die zur Berechnung von Teilrouten zur Verfügung stehen.
- `INTERACTION`
  Speichert die möglichen Interaktions-Typen zu einer externen Applikation (hier die Wrapper).
- `IA_INSTANCES`
  Speichert die von den externen Applikationen erstellten Interaktions-Instanzen und Interaktionsdaten.

- ALTER_VALUES
  Speichert die Alternativvorschläge für Haltestellenbezeichnungen zu einer Interaktions-Instanz.
- HAFAS
  Enthält die in der Fahrplanauskunft Hafas gefundenen Verkehrsverbindungen zu einer Teilroute.
- GRUPPE_TO
  Enthält die in der Flugauskunft Traveloverland gefundenen Fluggruppen zu einer Teilroute.
- VERBINDUNG_TO
  Enthält die in der Flugauskunft Traveloverland gefundenen Flugverbindungen zu einer Fluggruppe.
- RUECKFLUGVERBINDUNG_TO
  Enthält die in der Flugauskunft Traveloverland gefundenen möglichen Rückflugverbindungen zu einer Flugverbindung.
- KONKRETER_FLUG_TO
  Speichert die Flugdetails zu einer Flugverbindung.

A.4 Die Programmkomponenten

Folgende externe Applikationen realisieren die Funktionen der Arbeitsabläufe.

- routen_ermitteln.pl
  Perl-Programm, das aus den Anfragedaten die möglichen Reiserouten und die dazugehörigen Teilrouten ermittelt.
- teilrouten_aufbereiten.pl
  Perl-Programm, das zu einer Reiseroute die berechenbaren Teilrouten ermittelt.
- ermittle_wrapper.pl
  Perl-Programm, das für eine Teilroute, den zur Ermittlung der Verkehrsverbindungen zu benutzenden Wrapper zurückgibt.
- wf_appl_hafas.pl
  Workflow-Applikation (Perl-Programm), die den Wrapper für Hafas an das Workflow-System anbindet und die notwendigen Daten zur Wrapper-Steuerung aufbereitet.

- `wf_appl_traveloverland.pl`
  Workflow-Applikation (Perl-Programm), die den zusammengesetzten Wrapper für Traveloverland an das Workflow-System anbindet und die notwendigen Daten zur Wrapper-Steuerung aufbereitet.

- `teilroutenberechnung_wrapper_abschliessen.pl`
  Perl-Programm, das eine Teilroute für einen Wrapper als abgeschlossen austrägt.

- `Anbindung_Hafas.java`
  Java-Programm, das den Wrapper Hafas zur Verkehrsverbindungsabfrage benutzt. Es versorgt den Wrapper mit den notwendigen Parametern, speichert gefundene Verbindungen in der Datenbank ab und löst Nutzerinteraktionen aus.

  - `Wrapper_Hafas.w4f`
    Aus der Wrapper-Spezifikation erzeugt der W4F-Compiler die Java-Klasse `Wrapper_Hafas.java`, die das Java-Programm `Anbindung_Hafas.java` einbindet. Die Klasse erledigt das Abfragen der Verkehrsverbindungen von der Datenquelle Hafas und das Abbilden auf die Zieldatenstruktur, die von der Klasse `Verbindung_Hafas` beschrieben wird.

- `Anbindung_TO.java`
  Java-Programm, das die Zusammenarbeit der einzelnen Wrapper für die Ergebnisseiten von Traveloverland steuert. Es versorgt die Wrapper mit den notwendigen Parametern, speichert gefundene Verbindungen in der Datenbank ab und löst Nutzerinteraktionen aus. Die benutzten Wrapper-Klassen sind nachstehend aufgührt.

  - `Wrapper_TO_Fluggruppen.w4f`
    Aus der Wrapper-Spezifikation erzeugt der W4F-Compiler die Java-Klasse `Wrapper_TO_Fluggruppen`. Die Klasse erledigt das Abfragen der Fluggruppen von der Datenquelle Traveloverland und das Abbilden auf die Zieldatenstruktur, die von der Klasse `Fluggruppe_TO` beschrieben wird.
  - `Wrapper_TO_Flugverbindungen.w4f`
    Aus der Wrapper-Spezifikation erzeugt der W4F-Compiler die Java-Klasse `Wrapper_TO_Flugverbindungen`. Die Klasse erledigt das Abfragen der Flugverbindungen von der Datenquelle Traveloverland und das Abbilden auf die Zieldatenstruktur, die von der Klasse `Flugverbindung_TO` beschrieben wird.

  - `Wrapper_TO_Rueckflugverbindungen.w4f`
    Aus der Wrapper-Spezifikation erzeugt der W4F-Compiler die Java-Klasse `Wrapper_TO_Rueckflugverbindungen`. Die Klasse erledigt das Abfragen der Rückflugverbindungen von der Datenquelle Traveloverland und das Abbilden auf die Zieldatenstruktur, die von der Klasse `Rueckflugverbindung_TO` beschrieben wird.
  - `Wrapper_TO_Flugdetails.w4f`
    Aus der Wrapper-Spezifikation erzeugt der W4F-Compiler die Java-Klasse `Wrapper_TO_Flugdetails`. Die Klasse erledigt das Abfragen der Flugdetails von der Datenquelle Traveloverland und das Abbilden auf die Zieldatenstruktur, die von der Klasse `Flugdetails_TO` beschrieben wird.
  - `Wrapper_TO_Tarifdetails.w4f`
    Aus der Wrapper-Spezifikation erzeugt der W4F-Compiler die Java-Klasse `Wrapper_TO_Tarifdetails`. Die Klasse erledigt das Abfragen der Tarifdetails von der Datenquelle Traveloverland und das Abbilden auf die Zieldatenstruktur, die von der Klasse `Fluggruppe_TO` mit beschrieben wird.

- `DB_Data.java`
  Hilfsklasse zur Realisierung der JDBC-Datenbankanbindung.

- `Datum.java`
  Hilfsklasse, die zur Berechnung des Ankunftsdatums aus Abfahrtsdatum und Reisedauer benutzt wird. Weiterhin stellt sie benötigte Konvertierungsfunktionen zwischen verschiedenen Datums- und Zeitformaten zur Verfügung.

- `hauptmenu.cgi`
  Dieses Perl-Programm realisiert die Anfrageschnittstelle der Anfragesicht. Es erzeugt das Anfrageformular, speichert die Anfrage in die Datenbank ab und löst das Initialereignis zum Start des Workflows *Anfrage stellen* aus.

- `interaction.cgi`
  Dieses Perl-Programm ist den Interaktionstypen zugeordnet und wird bei Anforderung einer Nutzerinteraktion gestartet. Es behandelt eine auftretende Interaktion mit der entsprechenden Routine, z. B. Auswahl von Alternativen oder Eingabe eines neuen Wertes für einen beanstandeten Wert.

- `ergebnisausgabe.cgi`
  Dieses Perl-Programm übernimmt die Ergebnisdarstellung zu einer Anfrage. Es zeigt die zu einer Reiseroute gefunden Verkehrsverbindungen an.

Schlägt eine Verbindungsanfrage fehl, wird eine entsprechende Fehlermeldung ausgegeben.

Anhang B

Screenshots der WWW-Datenquellen

Abbildung B.1: Anfrageformular der Fahrplanauskunft Hafas der Deutschen Bahn

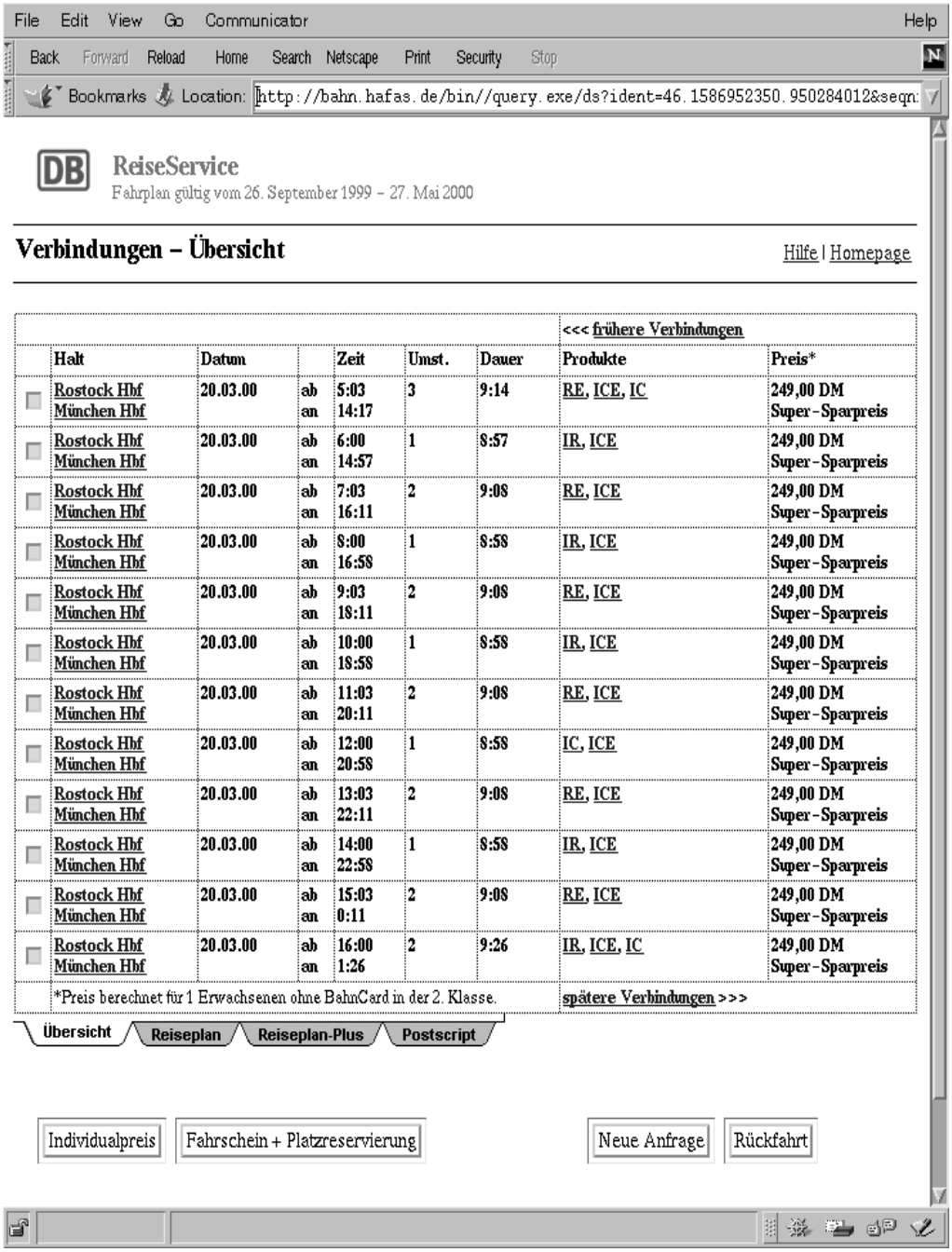

Abbildung B.2: Beispiel für ein Anfrageergebnis der Fahrplanauskunft Hafas der Deutschen Bahn

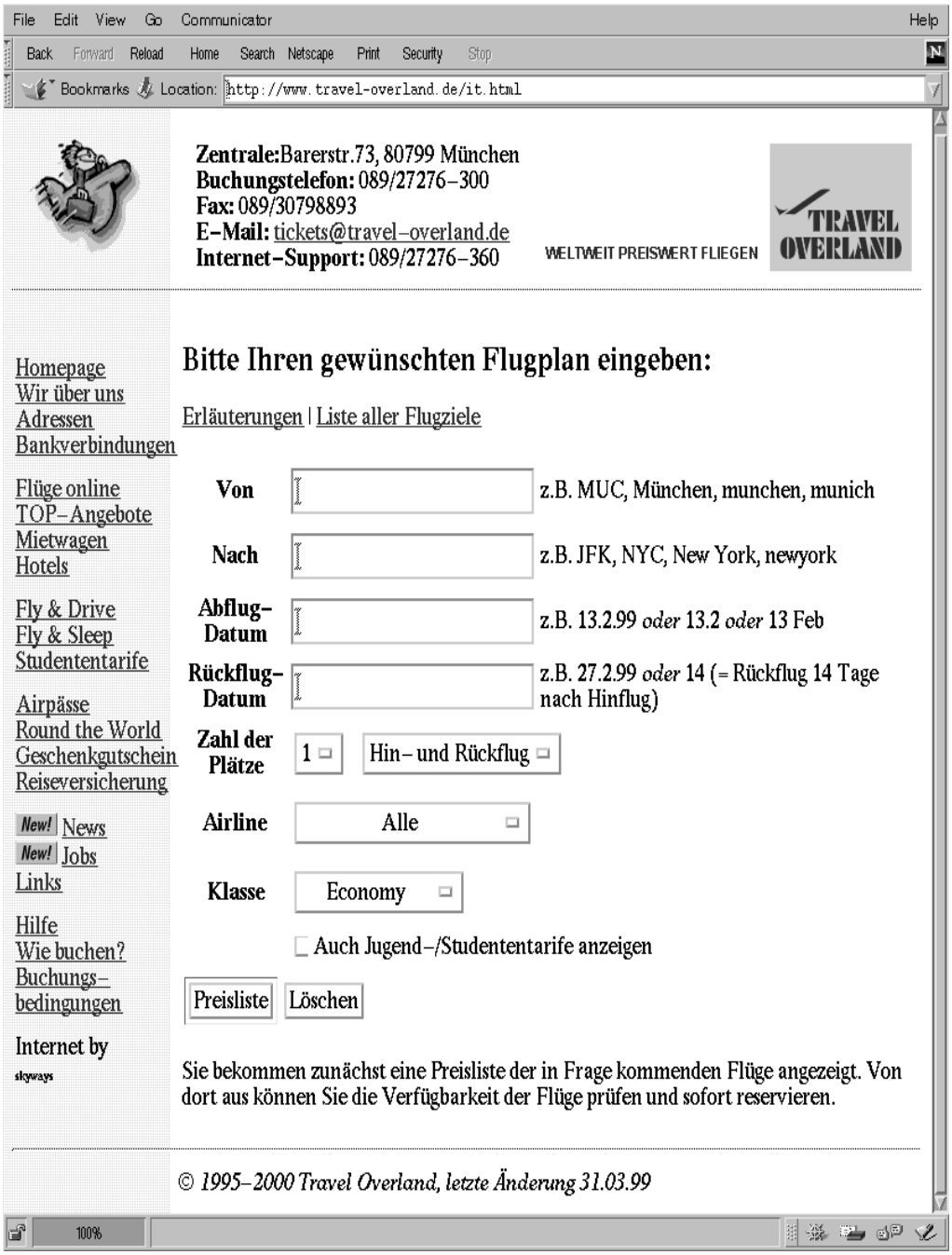

Abbildung B.3: Anfrageformular des Fluganbieters Traveloverland

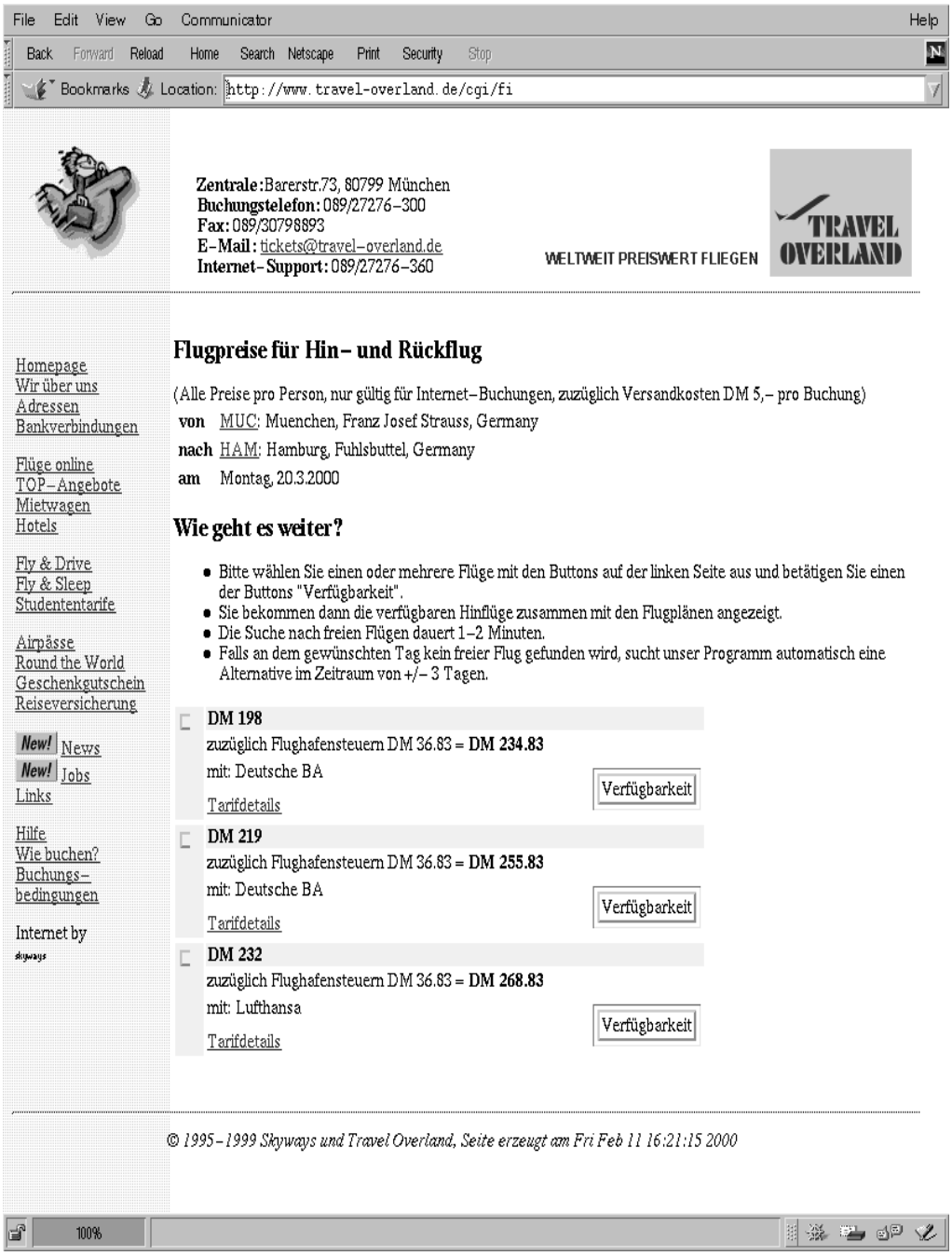

Abbildung B.4: Traveloverland-Beispiel: Erste Ergebnisseite zur Wahl einer Flugkategorie

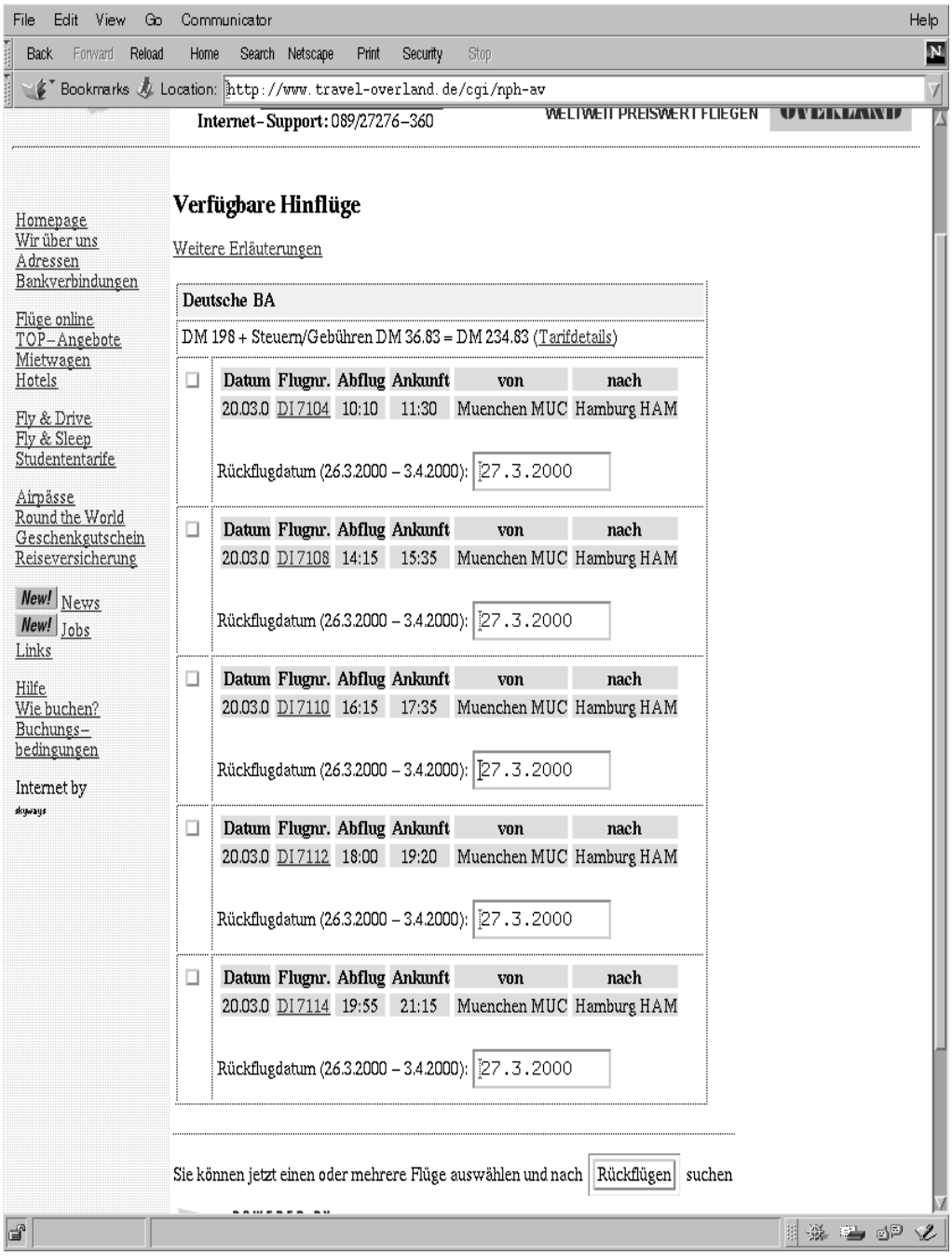

Abbildung B.5: Traveloverland-Beispiel: Nächste Ergebnisseite zur Wahl einer Flugverbindung

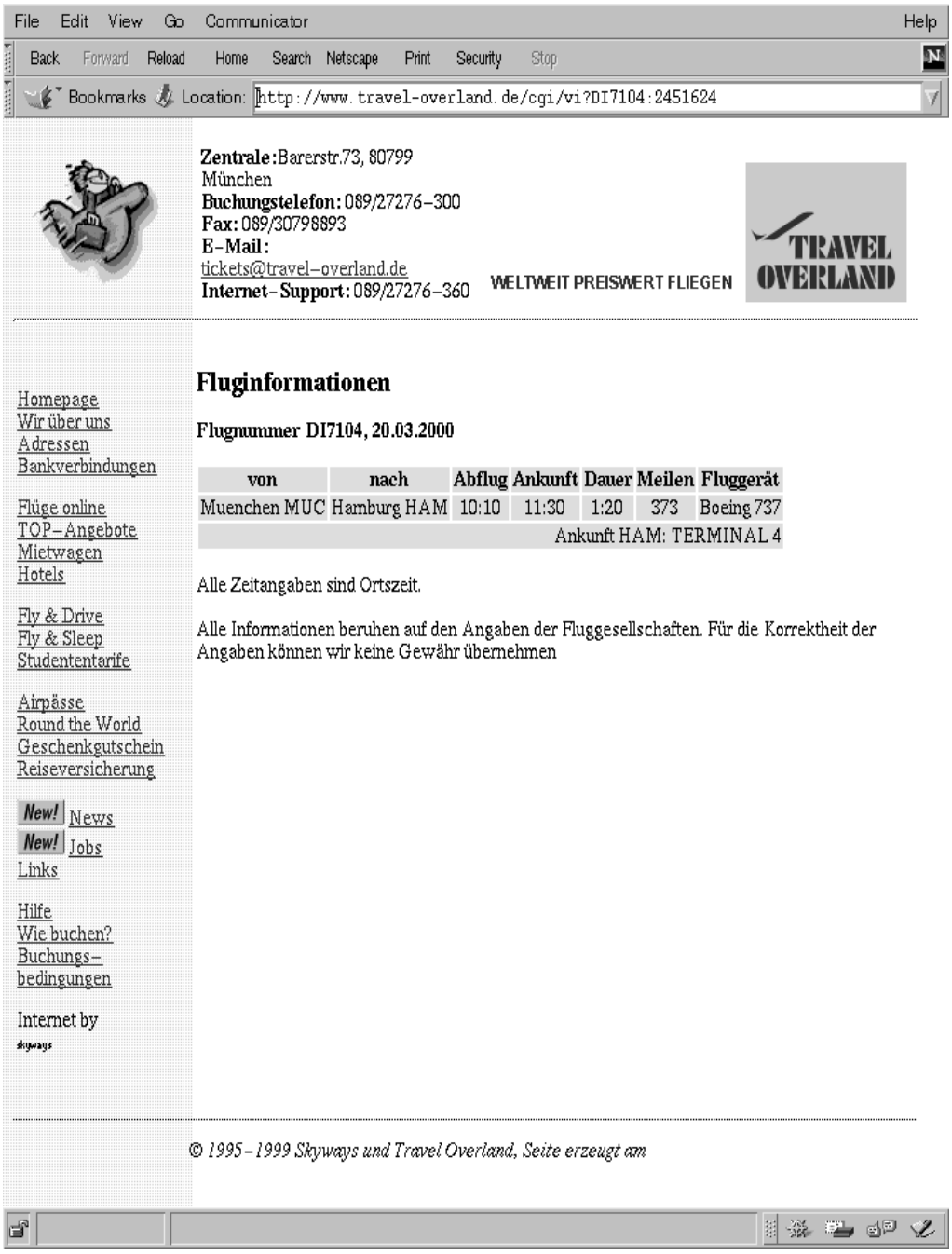

Abbildung B.6: Traveloverland-Beispiel: Ergebnisseite mit den Flugdetails einer ausgewählten Flugverbindung

Anhang C

Dokumentation des Workflow-Management-Systems

Als zugrundeliegendes DBMS wird *OpenIngres*[1] von *Computer Associates* benutzt. Es ist geeignet für diesen Einsatzzweck, da es aktive Datenbankmechanismen wie Trigger und Datenbankereignisse bietet. Leider ist die Konsistenzsicherung von Fremdschlüsselbeziehungen mit den SQL-92-Sprachkonstrukten `ON UPDATE`, `ON DELETE` in OpenIngres immer noch nicht möglich. Deswegen wurde zur Abbildung dieser Beziehungen über Trigger Holger Meyers `refingres`-Skript [Mey] benutzt. Implementiert wurde das Workflow-Management-System mit der Programmiersprache *Perl 5*[2] unter Benutzung des Perl-Moduls *DBI* (Database Interface) als Datenbankschnittstelle und dem Ingres-Datenbanktreiber.

C.1 Datenbankereignisse in OpenIngres

OpenIngres stellt ein Ereigniskonzept als aktiven Datenbankmechanismus zur Verfügung, mit dessen Hilfe Datenbankapplikationen über das DBMS Nachrichten austauschen können. Zu jeder Datenbank lassen sich Ereignisse unter einem bestimmten Namen anlegen (`CREATE DBEVENT event_name`). Eine Datenbankapplikation, die am Auftreten von bestimmten Ereignissen interessiert ist, teilt dies dem DBMS mit. Das geschieht mit dem SQL-Statement `REGISTER DBEVENT event_name`. Nach dieser Initialisierungsphase wartet die Applikation auf Meldung eines Ereignisses durch das DBMS. Dazu dient der Befehl `GET DBEVENT`. Dieser Befehl ist blockierend, d. h. es wird solange gewartet, bis ein Ereignis

[1] siehe OpenIngres-Handbücher [ing95b] und [ing95a]

[2] Eine Beschreibung der Sprache gibt es in [Sch95] und [WCS97].

auftritt. Durch die Angabe einer Zeitspanne kann die maximale Wartezeit festgelegt werden. Mit dem Statement `INQUIRE_SQL` lassen sich nach Auftritt eines Ereignisses der Ereignisname und eventuelle Parameter auslesen. Ein Ereignis kann man mit dem SQL-Statement `RAISE DBEVENT event_name` auslösen. Dabei ist es möglich, einen Ereignisparameter (bei OpenIngres maximal 255 Zeichen) anzugeben. Mit `REMOVE DBEVENT event_name` kann eine Applikation das Interesse an einem Ereignis beim DBMS abmelden und mit `DROP DBEVENT event_name` wird ein Ereignis aus der Datenbank gelöscht. OpenIngres speichert alle existierenden Ereignisse einer Datenbank in der Systemtabelle `iievents` und kümmert sich selbständig um die Verwaltung und Verteilung aufgetretener Ereignisse.

C.2 Datenbankprogrammierung mit Perl und DBI

Perl mit dem DBI-Package ist gut zur Datenbankprogrammierung geeignet. Durch die einheitliche Datenbankzugriffs-Schnittstelle DBI soll das Programm unabhängig von der konkreten Datenbank gemacht werden, die Zugriffsfunktionen zur Datenbank realisiert das entsprechende DBD-Modul (Data Base Driver). Nachfolgend sind die wesentlichen Zugriffsfunktionen aufgeführt.

- Datenbank öffnen:
```
$dbi_handle = DBI->connect($datasource, $std_dbuser,
$std_dbpass, { PrintError => 0, AutoCommit => 0 })
```
- SQL-Kommando ausführen:
```
$dbi_handle->do($sql_statement);
```
- Datenbankcursor anlegen:
```
$sql_cursor=$dbi_handle->prepare($sql_statement);
$sql_cursor->execute;
$row=$sql_cursor->fetch;
$sql_cursor->finish;
```
- Datenbank schließen:
```
$dbi_handle->commit;
$dbi_handle->disconnect;
```

C.3 Die Datenbankrelationen

Die Daten des Workflow-Management-Systems und die Daten des Beispielszenarios *Reiseverbindungen* werden in der OpenIngres-Datenbank `verbindung`

gespeichert. Eine Trennung in zwei unabhängige Datenbanken wäre wegen der Übersichtlichkeit besser, aber für eine prototypische Implementierung nicht unbedingt notwendig. Folgende Relationen nehmen die definierten Workflow-Schemata, Rollen, Initialereignisse und Operationstypdefinitionen auf:

- `WFTYPE`
  Enthält die Namen der definierten Workflow-Schemata.
- `WFTYPEATTRIBUTE`
  Speichert die Workflow-Konstantendefinition des Workflow-Schemas.
- `WFIATTRIBUTE`
  Speichert die Workflow-Variablendefinition des Workflow-Schemas.
- `WFIOPERATION`
  Speichert die Workflow-Operationsdefinition des Workflow-Schemas.
- `CF`
  Speichert die Kontrollflußdefinition des Workflow-Schemas.
- `WFTYPE_ROLE`
  Speichert die zum Workflow-Schema zugeordnete Rolle.
- `SUBWF`
  Speichert die Subworkflow-Definition des Workflow-Schemas.
- `STARTS`
  Speichert das zum Workflow-Schema zugeordnete Initialereignis.
- `DESCR_WFTYPE`
  Speichert den Workflowtyp-Aliasnamen und die Beschreibung des Workflow-Schemas.
- `DERIVED_OPERATION`
  Speichert die Zuordnung der abgeleiteten Operationen des Workflow-Schemas zu den Operationstypen.
- `OPERATIONTYPES`
  Enthält die Definitionen der Operationstypen.
- `INIT_EVENT`
  Enthält die Definitionen der Initialereignisse.
- `ROLES`
  Enthält die Rollendefinitionen.

Zur Mitglieder- und Benutzerverwaltung werden folgende Relationen benutzt:

- `PERSON`
  Enthält die Anwender des Workflow-Management-Systems.
- `PLAYS_ROLE`
  Speichert die zu den Anwendern zugeordneten Rollen, also die Berechtigungen.
- `ORGANIZATION`
  Enthält die existierenden Organisationen.
- `ORG_STRUCTUR`
  Speichert die Hierarchie zwischen den Organisationen *(nur vorgesehen, aber nicht verwendet).*
- `MEMBERSHIP`
  Speichert die Mitgliedschaften der Personen in den Organisationen, also die Mitglieder, und deren Position in der Organisation.

Die Workflow-Instanzen und deren Daten werden in diesen Relationen abgelegt:

- `WFI`
  Speichert die laufenden Workflow-Instanzen, mit dem Initiator, dem aktuellen Workflow-Status und dem Datum von dessen letzter Änderung sowie der Prozeßnummer des den Workflow ausführenden Prozesses. Der Workflow-Status wird von einem Trigger überwacht, der bei dessen Änderung ein Ereignis mit der Workflow-Instanznummer als Parameter auslöst. Das Datum der letzten Änderung und die Prozeßnummer werden zum Löschen von nicht beendeten Workflow-Instanzen benutzt.
- `WFIVARIABLES`
  Speichert die Workflow-Variablen und die internen Steuervariablen zu den Workflow-Instanzen. Die benutzten Steuervariablen sind:
  - `_address` speichert die Adresse zur Verbindung über Sockets.
  - `_initiator` speichert den Workflow-Ausführenden.
  - `_organisation` speichert die Organisation des Initiators.
- `APPL_DATA`
  Speichert Steuerparameter der CGI-Programme unter einem eindeutigen Schlüssel, um diese nach außen unsichtbar zu halten. Zusätzlich wird das Einfügedatum gespeichert, um nicht mehr benötigte Schlüssel zum Löschen zu erkennen.

C.4 Die Programmmodule

Das Workflow-Management-System setzt sich aus folgenden Hauptteilen zusammen:

- `eventmanager.pl`
  Der Ereignismanager ist die zentrale Komponente, siehe Abschnitt 4.2.

- `wf_new.pl`
  Das Instanziierungsmodul erstellt eine Workflow-Instanz zu einem Workflow-Schema. Es wird vom Ereignismanager und vom Interpretermodul zur Erstellung von Workflow-Instanzen aufgerufen. Die Parameterübergabe erfolgt über die Kommandozeile.

- `wf_execute.pl`
  Mit dem Start des Interpretermoduls beginnt die Ausführung der in der Kommandozeile übergebenden Workflow-Instanz.

- `Operation.pm`
  Das Operationsmodul wird zum Start von Operationen benutzt. Es kümmert sich um den Start der externen Programme und um die Kommunikation mit diesen.

- `Wf.pm`
  Dieses Modul stellt die Klasse *Workflow* mit deren Methoden (Erstellen, Ausführen, Verwaltung der Workflow-Daten ...) bereit.

- `Controlflow.pm`
  Dieses Modul dient zum Parsen des Kontrollflusses eines Workflows und der Ausführung der gefundenen Anweisungen.

- `clean_up.pl`
  Dient zum Aufräumen in den Relationen `WFI` (Workflow-Instanzen) und `APPL_DATA` (CGI-Steuerparameter). Tupel, deren eingetragenes Datum ein bestimmtes Alter überschreitet, werden gelöscht, da diese nicht mehr benötigt werden. Außerdem wird den zu den Workflow-Instanzen korrespondierenden Prozessen ein KILL-Signal geschickt, um diese Prozesse zu beenden.

- `time_control.pl`
  Dieses Skript wird vom Cron-Demon alle 24 Stunden ausgeführt. Es löst die Initialereignisse der Zeitüberwachungs-Workflows aus, die zeitabhängige Arbeiten automatisch erledigen.

Zum Einbringen der Definitionen in die Datenbank gibt es folgende Administrationswerkzeuge:

- `store_roles.pl`
  Die Rollendefinitionen werden aus einer in der Kommandozeile angegebenen Datei ausgelesen und in der Datenbank gespeichert.

- `store_init_events.pl`
  Die Definitionen der Initialereignisse werden aus einer in der Kommandozeile angegebenen Datei ausgelesen und in der Datenbank gespeichert.

- `store_operationtypes.pl`
  Die Definitionen der Operationstypen werden aus einer in der Kommandozeile angegebenen Datei ausgelesen und in der Datenbank gespeichert.

- `store_wf_definition.pl`
  Die Workflow-Schemata werden aus den in der Kommandozeile angegebenen Dateien ausgelesen und in der Datenbank gespeichert. Eine Datei darf nur ein Workflow-Schema enthalten.

Deweiteren werden diverse Hilfsmodule benutzt, die benötigte Funktionalitäten bereitstellen:

- `DB_dbi.pm`
  Klasse, die die Datenbankanbindung bereitstellt und Zugriffsfunktionen für SQL bietet.

- `Interaction.pm`
  Diese Modul dient zur Anbindung von CGI-Programmen an die WWW-Schnittstelle.

- `Hash.pm`
  Klasse zur Verwaltung von Schlüssel/Wert-Paaren.

- `List.pm`
  Klasse zur Realisierung des Listen-Datentyps der Workflow-Variablen.

- `Parameters.pm`
  Diese Klasse ist eine Erweiterung der Klasse Hash um Methoden zur Generierung von Zeichenketten für Parameterübergaben in der Kommandozeile, an Datenbankereignisse und an CGI-Programme.

- `Server.pm`, `Server_socket.pm`, `Client.pm`, `Client_socket.pm`, `Socket_user.pm`, `Stream.pm`
  Klassen zur Kommunikation über Sockets.

- `Query.pm`
  Klasse zur Verwaltung von CGI-Programmparametern und Nutzerautorisierung bei CGI-Programmen.

- `Html.pm`
  Klasse zum Erstellen von HTML-Dokumenten.

- `Table_info.pm`, `Attribute_info.pm`
  Klassen, die Informationen über Tabellen und Attribute der Datenbank liefern.

- `Worklist.pm`
  Klasse, die abhängig von den Berechtigungen eines Mitgliedes eine Liste von Workflow-Namen liefert.

- `misc.pm`
  Das Modul enthält allgemein benötigten Prozeduren, so z. B. für die Anbindung von externen Programmen an das Workflow-Management-System (Workflow-Applikationen).

C.5 Installation des Systems

Das Archiv `dawf.tar` wird mit nachstehendem Kommando extrahiert.

```
tar -xvf dawf.tar
```

Es wird das Verzeichnis `dawf` mit den Unterverzeichnissen `definitions`, `modules` und `extern` angelegt.

Nun muß als erstes die Datenbank angelegt werden. Dies erfolgt mit dem Kommando:

```
createdb verbindung
```

Wechseln Sie in das Unterverzeichnis `definitions`. Nun müssen die Tabellen in der Datenbank erstellt werden. Die dazu notwendigen SQL-Anweisungen generiert man mit dem `refingres`-Skript[3].

```
refingres verbindung.ref > verbindung.sql
```

[3] Siehe [Mey].

In der Datei `verbindung.ref` sind die Tabellen, die Primärschlüssel und die Fremdschlüsselbeziehungen definiert. Die SQL-Anweisungen aus der Datei `verbindung.sql` werden ausgeführt durch das Kommando:

```
sql verbindung < verbindung.sql
```

Die Zugriffspfade für die Tabellen sind in der Datei `verbindung_storage_structure.sql` festgelegt:

```
sql verbindung < verbindung_storage_structure.sql
```

Als nächstes erfolgt die Installation von notwendigen Triggern und Systemdaten:

```
sql verbindung < init_verbindung.sql
```

Nun müssen die Initialereignisse in der Datenbank angelegt und gespeichert werden:

```
store_init_events.pl init_events.def
```

Dann folgen die Rollen- und Operationstypdefinitionen:

```
store_roles.pl roles.def
```

```
store_operationtypes.pl operationstypes.def
```

Jetzt folgen die Workflow-Definitionen:

```
store_wf_definition.pl *.wf
```

Zum Schluß werden Initialisierungsdaten, die für die Benutzung des Systems notwendig sind, in die Datenbank eingetragen. Sie stehen in der Datei `init_nutzung.sql` und müssen eventuell per Hand an die Erfordernisse angepaßt werden.

```
sql verbindung < init_nutzung.sql
```

Weiterhin muß der Eintrag des Cron-Jobs erfolgen. In der Datei `cron_entry` ist der Eintrag gespeichert. Eine Anpassung des Zugriffspfades auf das auszuführende Programm ist erforderlich.

```
crontab cron_entry
```

Wechseln Sie in das Verzeichnis `modules`.

Mit dem Start des Ereignismanagers können Workflows ausgeführt werden.

```
eventmanager.pl &
```

Die zur Workflow-Ausführung nötigen externen Programme liegen im Verzeichnis `extern`. Im Unterverzeichnis `cgi-bin` liegen die CGI-Programme. Über einen symbolischen Link `modules` auf das Verzeichnis `dawf/modules` greifen die externen Programme auf die von ihnen benötigten Module zu. Der symbolische Link muß angepaßt werden, falls die externen Programme woanders liegen sollen.

Abbildungsverzeichnis

Literaturverzeichnis

[AL91] Appelrath, H.-J.; Ludewig, J.: *Skriptum Informatik - eine konventionelle Einführung*. B. G. Teubner, Stuttgart, 1991.

[Bul93] Bull S.A.: *Integration and Programming Guide FlowPATH*, 1993.

[Con97] Conrad, S.: *Föderierte Datenbanksysteme: Konzepte zur Datenintegration*. Springer Verlag, Berlin, 1997.

[Dic97] Dicken, H.: *JDBC - Internet-Datenanbindung*. International Thomson Publishing GmbH, Bonn, 1. Auflage, 1997.

[Fla98] Flanagan, D.: *Java Examples in a Nutshell*. O'Reilly Verlag, Köln, 1. Auflage, 1998.

[Goh99] Gohla, R.: Realisierung einer Studien- und Diplomarbeitsverwaltung mit Zugriff über das WWW mit Workflow-Mitteln. Studienarbeit, Lehrstuhl für Datenbank- und Informationssysteme des Fachbereiches Informatik der Universität Rostock, 1999.

[HFAN98] Huck, G.; Frankhauser, P.; Aberer, K.; Neuhold, E.: Jedi: Extracting and Synthesizing Information from the Web, 1998. Download unter URL http://www.darmstadt.gmd.de/oasys/projects/jedi/index.html.

[HS95] Heuer, A.; Saake, G.: *Datenbanken Konzepte und Sprachen*. International Thomson Publishing GmbH, Bonn, 1. Auflage, 1995.

[IBM96] IBM Deutschland Entwicklungsgesellschaft mbH, Wien: *FlowMark*, 3. Auflage, 1996.

[ing95a] *CA-OpenIngres - SQL Reference Guide*, 1995.

[ing95b] *CA-OpenIngres System Reference Guide*, 1995.

[JB96] Jablonski, S.; Bußler, C.: *Workflow-Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[JBS97] Jablonski, S.; Böhm, M.; Schulze, W.: *Workflow-Management Entwicklung von Anwendungen und Systemen*. dpunkt - Verlag für digitale Technologien GmbH, Heidelberg, 1. Auflage, 1997.

[jed] JEDI, Java Extraktion und Dissemination von Information. URL http://www.darmstadt.gmd.de/oasys/projects/jedi/index.html. Projektbeschreibung.

[Mar99] Marais, H.: WEBL - A Programming Language for the Web, 1999. Download unter URL http://www.research.digital.com/SRC/WebL/.

[Mey] Meyer, H.: refingres. AWK-Skript zur Erzeugung der referentiellen Integrität für Primär- und Fremdschlüssel in Ingres, aus der Vorlesung Datenbanken II des Fachbereiches Informatik der Universität Rostock, Lehrstuhl Datenbank- und Informationssysteme.

[MK97] Musciano, C.; Kennedy, B.: *HTML - The Definitve Giude*. O´Reilly Associates, Inc., Köln, 1. Auflage, 1997.

[Rug97] Rugama, T. R.: Integration von Workflow-Management-Funktionalität in eine CORBA-basierte CSCW-Umgebung. Diplomarbeit, Lehrstuhl für Datenbank- und Informationssysteme des Fachbereiches Informatik der Universität Rostock, 1997.

[SA98a] Sahuguet, A.; Azavant, F.: W4F: a WysiWyg Web Wrapper Factory for Minute-Made Wrappers, 1998. Download unter URL http://db.cis.upenn.edu/W4F/.

[SA98b] Sahuguet, A.; Azavant, F.: WysiWyg Web Wrapper Factory (W4F), 1998. Download unter URL http://db.cis.upenn.edu/W4F/.

[SBMW+96] Schulze, W.; Böhm, M.; Meyer-Wegener, K. et al.: *Services of Workflow Objects and Workflow Meta-Objects in OMG-compilant Enviroments, OOPSLA'96, Workshop on Business Object Design and Implementation*. San Jose, CA, 1996.

[Sch95] Schwartz, R. L.: *Einführung in Perl*. O'Reilly/International Thomson Verlag, Bonn, 1. Auflage, 1995.

[w4f] W4F's BNF. URL http://db.cis.upenn.edu/W4F/. Grammatik der Spezifikationssprachen eines W4F-Wrappers.

[WCS97] Wall, L.; Christiansen, T.; Schwartz, R. L.: *Programmieren mit Perl*. O'Reilly Verlag, Köln, 1. Auflage, 1997.

[Wor96] Workflow-Management Coalition, Bruessel: *Terminology & Glossary*, 1996.

Erklärung

Ich erkläre, daß ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 18.02.2000 Rolf Gohla

Thesen

I. Integrierte Anfragesichten können zu einer enormen Komfortsteigerung führen, da dem Nutzer lästige Arbeit bei der Suche nach Informtionen zu einem Thema abgenommen wird. Dabei ist die Qualitiät der Einbindung der WWW-Datenquellen entscheidend, damit nicht wesentliche Informationen verloren gehen.

II. Eine integrierte Anfragesicht kann neue Nutzerkreise erschließen, da nicht jeder Nutzer in der Lage ist, die von der Anfragesicht abgefragten und aufbereiteten Informationen selbst zu ermitteln.

III. Andere Darstellungsformen wie Multimedia-Elemente und Programme (Java) verdrängen allmählich die Seitenbeschreibungssprache HTML als Präsentationsmittel von Inhalten im WWW. Dadurch wird der Einsatzbereich für HTML-Wrapper immer kleiner.

IV. Zur Erreichung von Interoperabilität im WWW muß eine WWW-Datenquelle zukünftig auch eine Maschinen-Maschinen-Schnittstelle bieten. XML könnte eine Lösung dafür sein.

V. Prinzipiell treten bei der Integration von WWW-Datenquellen dieselben Probleme wie bei föderierten Datenbanksystemen auf.

VI. Die Arbeitsvorgangs- und Workflowmodellierung erzwingt eine klare Strukturierung und eine genaue Ablaufbeschreibung der Arbeitsaufgaben. Die dadurch erreichte Modularität sichert die Austauschbarkeit von Komponenten und die Erweiterbarkeit des Systems.

VII. CA-OpenIngres ist als Basis für die Implementierung eines WfMS geeignet. Die zur Verfügung stehenden aktiven Datenbankmechanismen wie Ereignisse und Trigger erweisen sich für die Realisierung der Workflow-Steuerung als sehr nützlich.